FernUniversität

Gesamthochschule Hagen

Fachbereich Informatik

Diplomarbeit

Zur Erlangung des Grades eines Diplom-Informatikers

November 1999

Visualisierung Geometrischer Datenstrukturen unter Berücksichtigung softwareergonomischer und lernpsychologischer Gesichtspunkte

geprüft von

Prof. Dr. Rolf Klein

Lehrgebiet Praktische Informatik VI

verfasst von

Ingrid Schumacher

und

Klaus-Dieter Schumacher

Zusammenfassung

Diese Diplomarbeit beschreibt unsere Intentionen für das entwickelte Programm und die Rahmenbedingungen für unsere objektorientierte Softwareentwicklung. Die Softwareentwicklung erfolgt angelehnt an den „Unified Software Development Process" von Booch, Jacobson und Rumbaugh [BOOC99]. Dabei setzen wir als Modellierungswerkzeug UML, die **U**nified **M**odeling **L**anguage, ein. Die Beschreibung einiger Schwierigkeiten bei der Implementation in Java und ein Ausblick schließen die Arbeit ab.

Das mit unserer Diplomarbeit entwickelte Softwareprodukt richtet sich vor allem an die Studierenden des Kurses Algorithmische Geometrie an der FernUniversität Hagen [FU1840]. Die im Kurs vorgestellten Datenstrukturen 2-dimensionaler Suchbaum und Prioritätssuchbaum werden mit dem Programm visualisiert.
So wird den Studierenden die Möglichkeit eröffnet, die Datenstrukturen durch Beobachtungen und selbständigen Aufbau zu analysieren und zu reflektieren. Dies geschieht durch das Nutzen der Demos und die Möglichkeit der experimentellen Auseinandersetzung mit den beiden Suchbäumen. Dabei lassen sich Aufbau, Abbau und Bereichsanfragen visualisieren und animieren. Durch das Programm werden die Datenstrukturen somit in klarerer Art und Weise dargestellt, als es in der Fachliteratur möglich ist, da diese nur Momentanaufnahmen in einzelnen Bildern festhalten kann.

I N H A L T S V E R Z E I CH N I S

1. AUFGABENSTELLUNG 1

2. EINGANGSVORAUSSETZUNGEN DER ADRESSATEN 4

3. RAHMENBEDINGUNGEN FÜR DIE SOFTWAREENTWICKLUNG 7

4. LERNPSYCHOLOGISCHE ASPEKTE 10

5. SOFTWAREERGONOMISCHE ASPEKTE 17

6. OBJEKTORIENTIERTE SOFTWAREENTWICKLUNG 24

7 ELABORATION / SYSTEMANALYSE UND ENTWURF 31

7.1 IST – UND SOLLZUSTAND 31
7.2 ARCHITEKTUR DES PROGRAMMS 34
7.3. GESTALTUNG DER BENUTZEROBERFLÄCHE 35
7.4 ELABORATION DES 2-D-BAUMS 37
7.4.1 Datenstruktur 37
7.4.1.1 Der ausgeglichene 2-d-Baum 38
7.4.1.2 Der allgemeine 2-d-Baum 38
7.4.1.3 Der dynamische 2-d-Baum 39
7.4.2 Use Cases zum 2-d-Baum 39
7.4.3 Verfeinerte Analyse mit Aktivitätsdiagrammen 44
7.4.3.1 Punkt in den dynamischen 2-d-Baum einfügen 44
7.4.3.2 Punkt aus dem dynamischen 2-d-Baum löschen 49
7.4.3.3 Punkte in den ausgeglichenen 2-d-Baum einfügen 55
7.4.3.4 Rechteckanfrage durchführen 58
7.4.3.5 Aufbau des allgemeinen 2-d-Baums 60
7.4.3.6 Demo aufrufen 61
7.4.3.7 Spezielles Menüsystem 62
7.5 ELABORATION DES PRIORITÄTSSUCHBAUMS 63
7.5.1 Datenstruktur 63
7.5.1.1 Der statische Prioritätssuchbaum 64
7.5.1.2 Der semidynamische Prioritätssuchbaum 64
7.5.1.3 Der dynamische Prioritätssuchbaum 65
7.5.1.4 Der balancierte dynamische Prioritätssuchbaum 65
7.5.2 Use Cases zum Prioritätssuchbaum 66

7.5.3 Verfeinerte Analyse mit Aktivitätsdiagrammen 68
7.5.3.1 Punkt in dynamischen Prioritätssuchbaum einfügen 69
7.5.3.2 Visualisieren des Einfügevorgangs 71
7.5.3.3 Punkt aus dem dynamischen Prioritätssuchbaum löschen 75
7.5.3.4 Visualisieren des Löschvorgangs 75
7.5.3.5 Bereichsanfrage durchführen 78
7.5.3.5 Demo einsetzen 80
7.5.3.7 Spezielles Menüsystem zum Prioritätssuchbaum 80
7.6 VORLÄUFIGE KLASSENBILDUNG 81

8. CONSTRUCTION / ERZEUGUNG 85

8.1 ITERATIVE UND INKREMENTELLE VORGEHENSWEISE 85
8.2 ENTWICKLUNG DER GEMEINSAMEN KOMPONENTEN 85
8.2.1 Entwicklung der Benutzeroberfläche 86
8.2.2 Algorithmus zum „schönen" Zeichnen der Suchbäume 92
8.2.3 Baumklassen 95
8.2.4 Drucken 97
8.2.5 Speichern und Laden der Punktmenge 98
8.2.6 Auswahl des Grafikformates 98
8.2.7 Das Java-Hilfesystem 99
8.3 CONSTRUCTION DES 2-D-BAUMS 103
8.3.1 Einfügen von Punkten im dynamischen 2-d-Baum 103
8.3.2 Bereichssuche im dynamischen 2-d-Baum 107
8.3.3 Löschen von Punkten im dynamischen 2-d-Baum 107
8.3.4 Der ausgeglichene 2-d-Baum 108
8.3.5 Schrittweiser Aufbau des ausgeglichenen 2-d-Baums 109
8.3.6 Der allgemeine 2-d-Baum 110
8.3.7 Animierter Aufbau des 2-d-Baums 116
8.3.8 Demos zum 2-d-Baum 116
8.4 CONSTRUCTION DES PRIORITÄTSSUCHBAUMS 117
8.4.1 Einfügen von Punkten im semidynamischen Baum 118
8.4.2 Einfügen von Punkten im dynamischen Baum 120
8.4.3 Bereichssuche im semidynamischen Baum 121
8.4.4 Löschen im semidynamischen und dynamischen Baum 122
8.4.5 Der semidynamisch-statische Baum 123
8.4.6 Der balancierte dynamische Baum 125
8.4.7 Erweiterung für die allgemeine Punktmenge 126
8.4.8 Animationen zum Prioritätssuchbaum 127
8.4.9 Demos zum Prioritätssuchbaum 132

9. PROGRAMMTEST 135

9.1 TESTS ZUM 2-D-BAUM 136
9.2 TESTS ZUM PRIORITÄTSSUCHBAUM 143

10. BEDIENUNGSANLEITUNG DES PROGRAMMS VISUALGEOM 152

10.1 SYSTEMVORAUSSETZUNGEN 152
10.2 INSTALLATION 154
10.3 EINSCHRÄNKUNGEN 155
10.4 STARTEN DER ANWENDUNG 156
10.5 DAS STARTMENÜ 158

10.6 DER 2-D-BAUM 159
10.6.1 Interpretation der Ausgaben zum 2-d-Baum 161
10.6.2 Das Menu Datei zum 2-d-Baum 164
10.6.3 Das Menü Bearbeiten zum 2-d-Baum 165
10.6.4 Das Menü Strukturvariante zum 2-d-Baum 165
10.6.5 Das Menü Optionen zum 2-d-Baum 167
10.6.6 Das Menü Hilfe zum 2-d-Baum 168
10.7 DER PRIORITÄTSSUCHBAUM 169
10.7.1 Interpretation der Ausgaben zum Prioritätssuchbaum 170
10.7.2 Das Menu Datei zum Prioritätssuchbaum 173
10.7.3 Das Menü Bearbeiten zum Prioritätssuchbaum 174
10.7.4 Das Menü Strukturvariante zum Prioritätssuchbaum 175
10.7.5 Das Menü Optionen zum Prioritätssuchbaum 176
10.7.6 Das Menü Hilfe zum Prioritätssuchbaum 177

11. SCHLUSSFOLGERUNGEN UND AUSBLICK 178

AUFTEILUNG DER GRUPPENARBEIT 180

ERKLÄRUNG 181

Verzeichnis der Abbildungen

Abbildung 1: Analyse des Battelle-Instituts 11
Abbildung 2: Repräsentationen nach Bruner 13
Abbildung 3: Der Software-Ergonomie zuzuordnende Themengebiete 18
Abbildung 4: Entwicklung objektorientierter Vorgehensmodelle 24
Abbildung 5: Der Rational Unified Process 25
Abbildung 6: 4+1-Sichten 29
Abbildung 7: Beispiel für die Baumdarstellung (Teil 1) 31
Abbildung 8: Beispiel für die Baumdarstellung (Teil 2) 32
Abbildung 9: Beispiel für die Baumdarstellung (Teil 3) 32
Abbildung 10: Beispiel für die Baumdarstellung (Teil 4) 33
Abbildung 11: Startbildschirm 35
Abbildung 12: Aufteilung des Arbeitsfensters in Bereiche 36
Abbildung 13: Use Case Diagramm 40
Abbildung 14: Koordinatentransformation 45
Abbildung 15: Aktivitätsdiagramm „Punkt in dynamischen 2-d-Baum einfügen“ 46
Abbildung 16: Beispielskizzen zum Einfügen des Punktes (6|9) 48
Abbildung 17: 2-d-Baum mit einem Blatt 50
Abbildung 18: 2-d-Baum mit zwei Blättern 50
Abbildung 19: 2-d-Baum mit zwei Blättern an einem Splitknoten 51
Abbildung 20: 2-d-Baum mit einen Blatt am Wurzelknoten 52
Abbildung 21: Löschen in einem beliebigen 2-d-Baum 53
Abbildung 22: 2-d-Baum nach Löschen des Punktes (10|7) 54
Abbildung 23: Aufbau eines ausgeglichenen 2-d-Baums (Teil 1) 55
Abbildung 24: Aufbau eines ausgeglichenen 2-d-Baums (Teil 2) 55
Abbildung 25: Aufbau eines ausgeglichenen 2-d-Baums (Teil 3) 56
Abbildung 26: Aufbau eines ausgeglichenen 2-d-Baums (Teil 4) 57
Abbildung 27: Aufbau eines ausgeglichenen 2-d-Baums (Teil 5) 57
Abbildung 28: Aktivitätsdiagramm „Anfragerechteck” 59
Abbildung 29: Beispielskizze zum Anfragerechteck 60
Abbildung 30: Menüsystem zum 2-d-Baum 62
Abbildung 31: Use Case Diagramm 66
Abbildung 32: Koordinatentransformation 69
Abbildung 33: Aktivitätsdiagramm „Punkt in Prioritätssuchbaum einfügen” 70
Abbildung 34: Animiertes Einfügen eines Punktes 72
Abbildung 35: Beispielskizzen zum animierten Einfügen in den dynamischen Baum 73
Abbildung 36: Balancieren im Prioritätssuchbaum (Teil 1) 74
Abbildung 37: Balancieren im Prioritätssuchbaum (Teil 2) 74
Abbildung 38: Animiertes Löschen im semidynamischen Prioritätssuchbaum 76
Abbildung 39: Animiertes Löschen im dynamischen Prioritätssuchbaum 77
Abbildung 40: Aktivitätsdiagramm „Bereichsanfrage” 78
Abbildung 41: Beispielskizze zur Bereichsanfrage 79
Abbildung 42: Spezielles Menüsystem zum Prioritätssuchbaum 80
Abbildung 43: Vorläufiges Klassendiagramm zum 2-d-Baum 83
Abbildung 44: Vorläufiges Klassendiagramm zum Prioritätssuchbaum 84
Abbildung 45: Basisfenster für die Suchbäume 87
Abbildung 46: Klassendiagramm zur Benutzeroberfläche 88
Abbildung 47: Zusammenstellen der Fensterkomponenten 89
Abbildung 48: Zusammenspiel der Klassen nach dem MVC-Prinzip 91
Abbildung 49: Interaktionsdiagramm zum Anmelden eines ChangeListeners 92
Abbildung 50: Schönes Zeichnen mit relativem Offset und berechnetem x-Wert 94

Abbildung 51: Klassendiagramm der implementierten Baumklassen....96
Abbildung 52: Druckbereich eines DIN A4-Blattes....97
Abbildung 53: Grundsätzlicher Aufbau des Hilfesystems....101
Abbildung 54: Nutzen des HelpSet aus einer Anwendung....102
Abbildung 55: Interaktionsdiagramm zum dynamisch-interaktiven 2-d-Baum....105
Abbildung 56: Knoten nicht im Zeichenbereich....106
Abbildung 57: schrittweiser Aufbau des ausgeglichenen 2-d-Baums....109
Abbildung 58: leere Restpunktliste....111
Abbildung 59: Restpunktliste mit einem Punkt....112
Abbildung 60: Restpunktliste mit zwei Punkten....112
Abbildung 61: Restpunktliste mit drei Punkten....113
Abbildung 62: Restpunktliste mit mehr als drei Punkten....113
Abbildung 63: unausgeglichene Form des 2-d-Baums....114
Abbildung 64: Klassendiagramm zum ternären 2-d-Baum....115
Abbildung 65: Einfügen von Punkten im semidynamischen Prioritätssuchbaum....118
Abbildung 66: Interaktionsdiagramm: Einfügen im semidynamischen Baum (Teil 1)....119
Abbildung 67: Interaktionsdiagramm: Einfügen im semidynamischen Baum (Teil 2)....120
Abbildung 68: Bereichsanfrage im semidynamischen Prioritätssuchbaum....121
Abbildung 69: Löschen eines Punktes im dynamischen Prioritätssuchbaum....123
Abbildung 70: Linksvollständige Baumskelette....123
Abbildung 71: Beispiel zum Punkteinfügen in den dynamisch balancierten Baum....125
Abbildung 72: Animierte Erweiterung des alten Baums....128
Abbildung 73: Animiertes Einfügen eines neuen Knotens....129
Abbildung 74: Animiertes Einfügen eines neuen Punktes....130
Abbildung 75: Animiertes Einfügen mit Verdrängen eines Punktes....130
Abbildung 76: Animiertes Löschen im dynamischen Prioritätssuchbaum....131
Abbildung 77: Interaktionsdiagramm zum „Zusammenspiel der Threads“....133
Abbildung 78: Startmenü der Applikation von **VisualGeom**....158
Abbildung 79: Startmenü des Applets von **VisualGeom**....158
Abbildung 80: Startfenster des 2-d-Baums....159
Abbildung 81: Animierter dynamischer 2-d-Baum....161
Abbildung 82: Anfrage durchführen....161
Abbildung 83: Schrittweiser Aufbau des ausgeglichenen 2-d-Baums....162
Abbildung 84: Allgemeiner 2-d-Baum (Teil 1)....162
Abbildung 85: Allgemeiner 2-d-Baum....163
Abbildung 86: Startfenster des Prioritätssuchbaums....169
Abbildung 87: Animierter Aufbau beim semidynamisch statischen Prioritätssuchbaum....170
Abbildung 88: Animiertes Punkt-Einfügen in den semidynamischen Prioritätssuchbaum....171
Abbildung 89: Animierter Aufbau beim dynamischen Prioritätssuchbaum....171
Abbildung 90: Animiertes Löschen eines Punktes....172
Abbildung 91: Löschen im dynamischen Prioritätssuchbaum....172
Abbildung 92: Anfrage durchführen....173

Verzeichnis der Tabellen

TABELLE 1: MAXIMALE KNOTENANZAHL DES ALLGEMEINEN 2-D-BAUMS 61
TABELLE 2: KLASSEN AUS DEM PAKET **COMMON** 96
TABELLE 3: FÜR DIE APPLIKATION BENÖTIGTE SOFTWARE 152
TABELLE 4: FÜR DAS APPLET BENÖTIGTE SOFTWARE 153
TABELLE 5: AUFRUF VON **VISUALGEOM** ALS APPLET 157
TABELLE 6: BEDEUTUNG DER VIER BUTTONS BEIM 2-D-BAUM 160
TABELLE 7: 2-D-MENÜPUNKTE DES DATEI-MENÜS 164
TABELLE 8: 2-D-MENÜPUNKTE DES BEARBEITEN-MENÜS 165
TABELLE 9: 2-D-MENÜPUNKTE DES MENÜS STRUKTURVARIANTE (TEIL 1) 166
TABELLE 10: 2-D-MENÜPUNKTE DES MENÜS STRUKTURVARIANTE (TEIL 2) 167
TABELLE 11: 2-D-MENÜPUNKTE DES MENÜS OPTIONEN 167
TABELLE 12: 2-D-MENÜPUNKTE DES HILFE-MENÜS 168
TABELLE 13: BEDEUTUNG DER DREI BUTTONS BEIM PRIORITÄTSSUCHBAUM 170
TABELLE 14: PRIO-MENÜPUNKTE DES DATEI-MENÜS (TEIL 1) 173
TABELLE 15: PRIO-MENÜPUNKTE DES DATEI-MENÜS (TEIL 2) 174
TABELLE 16: PRIO-MENÜPUNKTE DES BEARBEITEN-MENÜS 174
TABELLE 17: PRIO-MENÜPUNKTE DES MENÜS STRUKTURVARIANTE (TEIL 1) 175
TABELLE 18: PRIO-MENÜPUNKTE DES MENÜS STRUKTURVARIANTE (TEIL 2) 176
TABELLE 19: PRIO-MENÜPUNKTE DES MENÜS OPTIONEN 176
TABELLE 20: PRIO-MENÜPUNKTE DES HILFE-MENÜS 177

Tell me and I forget,
Teach me and I remember,
Involve me and I learn.

Benjamin Franklin[RICH89]

1. Aufgabenstellung

Mit dieser Diplomarbeit haben wir vom Lehrgebiet Praktische Informatik VI den Auftrag erhalten, ein Java-Programm zur Visualisierung der geometrischen Datenstrukturen 2-dimensionaler Suchbaum und Prioritätssuchbaum zu erstellen. Beide Bäume sind Beispiele für Datenstrukturen, die in der algorithmischen Geometrie genutzt werden, um effizient Punkte in der Ebene zu speichern. Sie unterstützen die Suche nach einzelnen Punkten und Bereichsanfragen.

Der 2-dimensionale Suchbaum

Der 2-dimensionale Suchbaum (2-d-Baum) stellt eine Erweiterung des eindimensionalen Suchbaums dar. Auf jeder Ebene des Suchbaums wird die nach der jeweiligen Koordinate sortierte Punktmenge zyklisch nach einer der beiden Koordinaten aufgeteilt. Diese wird auch als Splitkoordinate bezeichnet. Im Wurzelknoten wird immer mit der ersten Koordinate begonnen. Hierdurch entstehen in der Ebene Splitgeraden. Sie teilen die Punktmenge auf. Wenn sich die Punkte in allen Koordinaten voneinander unterscheiden, differieren die beiden so entstandenen Punktteilmengen maximal um eins.

„Ein ausgeglichener 2-d-Baum für n Punkte in der Ebene lässt sich in Zeit $O(n \log n)$ konstruieren. Er belegt $O(n)$ viel Speicherplatz. Eine Bereichsanfrage mit achsenparallelem Rechteck lässt sich in Zeit $O(\sqrt{n} + a)$ beantworten; hierbei bezeichnet a die Größe der Antwort.“ [FU1840, S. 121]

Der Prioritätssuchbaum

Der Prioritätssuchbaum dient ausschließlich zur Speicherung von Punkten in der Ebene. Er besteht aus einer Kombination eines eindimensionalen Bereichsbaums und eines Punkte-Heaps. Beim eindimensionalen Bereichsbaum handelt es sich um einen ausgeglichenen binären Suchbaum. Es wird zwischen dynamisch und statisch aufgebauten Prioritätssuchbäumen unterschieden.

Vor dem Aufbau eines statischen Prioritätssuchbaums muss die Punktemenge bekannt sein. Es erfolgt erst der Aufbau eines eindimensionalen Bereichsbaums. In diesen werden anschließend die gegebenen Punkte unter Berücksichtigung folgender Kriterien eingetragen:

- Jeder Punkt befindet sich auf dem Suchpfad zu seiner x-Koordinate.
- Die Werte der y-Koordinaten steigen entlang des Pfades von der Wurzel zu einem Blatt monoton.
- Die Punkte sind so im Prioritätssuchbaum zu platzieren, dass sie möglichst dicht an der Wurzel stehen.

„Ein Prioritätssuchbaum für n Punkte in der Ebene kann in Zeit $O(n \log n)$ aufgebaut werden. Er benötigt $O(n)$ viel Speicherplatz und gestattet es, eine Halbstreifenanfrage in Zeit $O(\log n + a)$ zu beantworten.“ [FU1840, S. 132]

Das Hauptziel des zu entwickelnden Programms stellt die Nutzbarkeit für die Studenten[1] dar. Ihnen soll ermöglicht werden, Schwierigkeiten bei der Auseinandersetzung mit den beiden geometrischen Datenstrukturen zu reduzieren und zu beseitigen.
Hierzu soll das Programm begleitend zur Einheit 3 des Kurses 1840 - Algorithmische Geometrie – die Möglichkeit eröffnen, mit den Datenstrukturen 2-d-Baum und Prioritätssuchbaum zu experimentieren:

[1] Zur besseren Lesbarkeit verwenden wir in unserer Diplomarbeit den Begriff Student im Sinne von „pars pro toto“.

- Im dynamischen Fall erfolgt nach der Eingabe eines Punktes die Änderung im jeweiligen Suchbaum.
- Im statischen Fall wird nach der Eingabe aller Punkte ein visualisierter Baumaufbau durchgeführt.
- Bei Bereichsanfragen wird dargestellt, welche Punkte gefunden wurden und welche Knoten hierfür im Baum angefasst werden mussten.

Um die Datenstrukturen sinnvoll nutzen zu können, soll eine einheitliche benutzerfreundliche Bedieneroberfläche erstellt werden. Sie soll auch für die unterschiedlichsten Anwendungen aus dem Bereich der algorithmischen Geometrie einsetzbar sein. Dabei sind die Möglichkeiten von JDK2 bei der Gestaltung von Benutzeroberflächen zu berücksichtigen.
Des weiteren wird der Einsatz der Software als Applikation und als Applet gewünscht.

2. Eingangsvoraussetzungen der Adressaten

Wesentlich für die Entwicklung eines Softwareproduktes ist das Ermitteln und Berücksichtigen der Eingangsvoraussetzungen der Benutzer. Zur Evaluation[2] wäre hierfür in unserer Diplomarbeit die Erhebung und Analyse einer statistisch gesicherten Stichprobe bei den Studierenden des Kurses Algorithmische Geometrie durchzuführen. Dies war leider nicht möglich, da sich keine zeitliche Übereinstimmung von Diplomarbeit und Kurs ergab.
Bei den weiteren Ausführungen greifen wir deshalb auf unsere eigenen Erfahrungen zurück. Diese haben wir zum einen beim Studium des Kurses Algorithmische Geometrie [FU1840] und der Vorbereitung auf die Diplomprüfung erworben. Hinzu kommen die Erkenntnisse aus der Betreuung von Studenten im Rahmen unserer Tätigkeit als Mentoren für Informatik am Studienzentrum in Lübeck.

Zu Beginn des Kurses Algorithmische Geometrie sollten die Studenten über Kenntnisse aus dem Kurs Datenstrukturen [FU1663] verfügen. Da der zeitliche Abstand zwischen beiden Kursen vor allem bei Teilzeitstudenten recht lang ist, muss berücksichtigt werden, dass diese Kenntnisse oft nicht mehr präsent sind.
Wir können also nur ein grundsätzliches Verständnis über Bäume und deren Aufbau voraussetzen. Einfügen, Löschen und Balancieren sind bekannte Begriffe, jedoch nicht so präsent, dass flüssig und verständig mit ihnen gearbeitet werden kann.

Die Studenten benötigen deshalb animierte Visualisierungen beim Aufbauen, Einfügen und Löschen der beiden Suchbäume. Des weiteren sollen sie möglichst erfahren, was das Entarten von Bäumen bedeutet und wie sich damit das Laufzeitverhalten verändert. Dabei ist die wesentliche mit dem Programm zu verfolgende Intention, dass die Studenten die im Kurstext beschriebenen beiden Suchbäume interaktiv erleben können.

[2] Geeignete Ausführungen zur Evaluation von Software finden sich unter [FU4763, Seite 86 bis 110].

Des weiteren setzen wir bei den Benutzern grundlegende Fähigkeiten und Fertigkeiten im Umgang mit Tastatur, Maus, Bildschirm und Anwendungssoftware voraus. Dies schließt auch die Fähigkeit ein, unser Programm einschließlich dem **J**ava **R**untime **E**nvironment (**JRE2**) oder dem **J**ava **D**evelopment **K**it (**JDK2**)[3] gemäß einer Anleitung[4] auf der Festplatte zu installieren.

Als zusätzliche Rahmenbedingung wollen wir berücksichtigen, dass die Studenten bis zu diesem Zeitpunkt ihres Studiums zumeist nur Erfahrungen im Umgang mit Microsoft-Oberflächen gemacht haben. Deshalb wollen wir uns bei der Gestaltung des Menüsystems daran orientieren.

Schlussfolgerung:

> Die Software soll den bisherigen Erfahrungen der Benutzer in
> - der Funktionalität,
> - den Eigenschaften und
> - dem Verhalten
>
> so weit als möglich entsprechen.

Bei dem späteren Einsatz des Programms gehen wir von zwei verschiedenen Benutzerklassen aus:

- **Primäre Benutzerklasse**
  Hierzu zählen wir Studenten oder sonstige Personen aus dem Fachbereich Informatik, die sich gezielt mit dem 2-d-Baum oder dem Prioritätssuchbaum beschäftigen wollen.

- **Sekundäre Benutzerklasse**
  Dieser Klasse ordnen wir u. a. die Mitarbeiter des Lehrgebietes zu, die das Programm zur Veranschaulichung bei Studientagen oder zum Erstellen von Einsendeaufgaben nutzen wollen.

[3] Dieses wird von SUN auch als Java 2 SDK, Version 1.2.x, Standard Edition bezeichnet. Wir verwenden im weiteren die Abkürzung JDK2.

[4] Siehe hierzu Kapitel 10.2.

Wir werden deshalb nachfolgend statt des Begriffes Student nur noch den Begriff Benutzer[5] verwenden.

Aus den beiden Benutzerklassen ergeben sich Konsequenzen für den Funktionsumfang und die Anforderungen an das von uns zu erstellende Programm.

[5] Auch den Begriff Benutzer verwenden wir im Sinne von „pars pro toto“.

3. Rahmenbedingungen für die Softwareentwicklung

Unser Programm soll sowohl als Applet als auch als Applikation lauffähig sein und dies möglichst auf den Betriebssystemen

- Windows in den Versionen 95, 98 und NT,
- Linux,
- Solaris und
- MAC-OS.

Programme in Sprachen wie C++ müssen nach der Portierung auf ein anderes Betriebssystem neu compiliert und ggf. an betriebssystemspezifische Besonderheiten angepasst werden. Die von der Firma SUN entwickelte Programmiersprache Java übersetzt beim Compilieren Quellcode in Bytecode. Dieser ist mit dem im JRE bzw. dem JDK enthaltenen Interpreter ohne Änderungen auf den oben genannten Betriebssystemen lauffähig.
Für uns ergibt sich damit zwangsläufig, das Programm in der Sprache Java zu implementieren. Nur mit dieser objektorientierten Programmiersprache ist die betriebssystemspezifische Plattformunabhängigkeit realisierbar.

Damit verbleibt aktuell nur noch die Auswahl zwischen den Versionen JDK1.1 und JDK2. Auf Wunsch des Lehrgebietes und um die unter JDK2 bereits integrierten Swing-Klassen[6] und das Java-2D-API[7] nutzen zu können, entschieden wir uns für JDK2. Hierdurch ergaben sich während der Programmentwicklung allerdings diverse Schwierigkeiten.
Zu Beginn der Diplomarbeit existierte nur eine Beta-Release von JDK2 unter Windows und unter Solaris. Die erste Pre-Release für Linux erschien erst im Frühjahr 1999. Sie setzt allerdings die Verwendung der glibc-Bibliotheken voraus. Erst ab der Version 6.0 sind diese ein Bestandteil der SuSE[8]-Distribution. Eine MAC-OS-Version von JDK2 existiert nach unserem Kenntnisstand immer noch nicht.

---

6 Hierbei handelt es sich um eine Erweiterung des Abstract Window Toolkit.

7 Hiermit stellt Java Grafikfunktionen für Text-, Grafik- und Bilddarstellungen zur Verfügung, die außerdem unabhängig von den physikalischen Geräten Drucker und Bildschirm sind.

8 SuSE GmbH, Schanzäckerstr. 10, D-90443 Nürnberg; http://www.suse.de/

Auch unterstützen die verfügbaren Browser JDK2-Applets nicht. Dies soll sich mit einer demnächst erscheinenden Version des Netscape-Communicators ändern. Dafür ist z. Z. ein Plug-In für den Netscape Communicator und den Internet Explorer Bestandteil der Windows-Version von JRE. Auch unter Solaris lassen sich seit August 1999 JDK2-Applets mit einem Plug-In nutzen. Zusätzlich ist aber für JDK2 ein umfangreicherer Betriebssystempatch erforderlich.
Für Linux existiert weiterhin kein Plug-In. Hier verbleibt nur die Nutzung des in JDK2 enthaltenen Programms **`appletviewer`**.

Für die Nutzung des Applets mit einem Browser muss die HTML-Seite, aus der das Applet aufgerufen wird, unter Verwendung einer speziellen Java-Applikation[9] aufbereitet werden. Erst dann kann der Browser unter Nutzung des JRE das Applet starten. Mit der so modifizierten HTML-Seite lässt sich das Applet allerdings nicht mit dem **`appletviewer`** aufrufen.

Als Fazit dieser Ausführungen bleibt, dass gegenwärtig bei der Nutzung als Applet zusätzlicher Aufwand notwendig ist. Dies stellt einen Nachteil dar, der unseres Erachtens bei der Nutzung von JDK2-Applets im Rahmen des Internets nicht zu gering eingeschätzt werden darf.

Weitere Java-Tools wie das Java-Hilfesystem **`JavaHelp`** befanden sich bis Juli 1999 noch in ihrer Entwicklung. **`JavaHelp`** ist kein Bestandteil des JDK. Die aktuelle Version des **`appletviewer`** ist laut SUN nicht dafür angepasst.
Das JPEG-Paket ist Bestandteil des JDK2. Nutzen konnten wir es unter den Versionen 1.2 und 1.2.1 jedoch erst, nachdem es compiliert, in eine **`jar`**-Datei[10] konvertiert und dann in den **`CLASSPATH`**[11] eingebunden wurde.

Die Situation der permanenten Fortentwicklung von Java gestaltete die Implementation der Software nicht leichter. Hinzu kam, dass die von SUN viel gerühmte völlige Plattformunabhängigkeit doch nicht gegeben ist (siehe hierzu auch Kapitel A.4 im Anhang).

[9] Java(tm) Plug-In HTML-Converter

[10] Hierbei handelt es sich um ein komprimiertes Java-Archiv, das mit dem **J**ava **Ar**chive Tool erzeugt wurde.

[11] Im dieses Umgebungsvariablen wird festgelegt, von wo die Systemklassen zu importieren sind.

Deshalb beschlossen wir, folgende Rahmenbedingungen festzulegen, um ein gewisses Maß an Stabilität in unsere Implementationsaktivitäten zu bekommen:

- Die Entwicklung erfolgt mit JDK2 unter Windows95 bzw. Windows98 mit Kawa[12] als integriertem Entwicklungswerkzeug. Ergänzend werden die Klassen von JavaHelp und das compilierte Paket **`com.sun.image.codec.jpeg`** genutzt.

- Unter Verwendung der zusätzlichen Klassen werden die von uns programmierten Klassen als Applikation unter Solaris mit JDK1.2 und unter Linux mit JDK1.2pre1 getestet.

- Die Applets werden unter Windows mit dem Netscape Communicator 4.5 und dem Internet Explorer in der Version 4 und unter Solaris und Linux mit dem **`appletviewer`** getestet.

[12] Dies ist eine integrierte unter „http://www.tek-tools.com/kawa“ erhältliche Entwicklungsumgebung.

4. Lernpsychologische Aspekte

Dieses Kapitel besitzt für uns eine doppelte Bedeutung:

- Als Reflexion über Lerntypen, Lernkanäle und Darstellungsarten repräsentiert es für uns eine wesentliche Voraussetzung für die Entwicklung des geplanten Softwareproduktes und die Formulierung der Anforderungen an das fertige Produkt.
- Es zeigt auf, welche Einsatzmöglichkeiten wir für unser Programm sehen.

Damit führen wir hier und in den Kapiteln 2, 3 und 5 auch eine Voruntersuchung[13] durch.

Sowohl die Theorie als auch die Praxis des Lehrens und Lernens ist ein wesentlicher Bestandteil unserer menschlichen Lebensweise. Dabei spielt das Bild, das wir uns vom Menschen machen, eine besondere Bedeutung. Die jeweiligen Ansätze der Lernpsychologie ergeben sich deshalb aus dem jeweils zugrunde gelegten Menschenbild.
Ausgehend vom betreffenden Menschenbild bietet die Psychologie verschiedene Modelle für die Erklärung des Zustandekommens von Lernprozessen im Lernenden an. Eines dieser Modelle geht vom handelnden Menschen aus. Dieser lernt durch sein Tun. Dabei werden durch den Einsatz geeigneter Verstärker Verhaltensweisen und Handlungen gefördert. Angemerkt sei, dass auch das Verfahren „Trial an Error"[14] bei positiver Rückmeldung eine Verstärkung herbeiführen kann.

Mit unserem Programm sollen folgende Möglichkeiten eröffnet werden:

- Das übliche Lehr-Lern-Konzept der Hochschule, bei dem die Stoffvermittlung im Rahmen einer systematisch geordneten Lehrbuchstruktur erfolgt, kann zumindest in einem Teilbereich des Kurses „Algorithmische Geometrie" [FU1840, Seiten 118 bis 125 und 130 bis 135] durchbrochen werden.
- Das Programm kann im Rahmen der virtuellen Universität durch geeignete Querverweise im Kurstext von [FU1840] gezielt eingesetzt werden.
- Die Benutzer setzen das Programm zur Lösung geeignet aufbereiteter Einsendeaufgaben zu [FU1840] ein.

[13] Siehe Ausführungen zur Inception-Phase / Voruntersuchung im Kapitel 6.
[14] Gerade die naive Vorgehensweise bei diesem Verfahren bietet sich beim Umgang mit dem Programm und bei Auswahl der Animation an und kann so zu grundlegenden Einsichten über den Aufbau der Suchbäume führen.

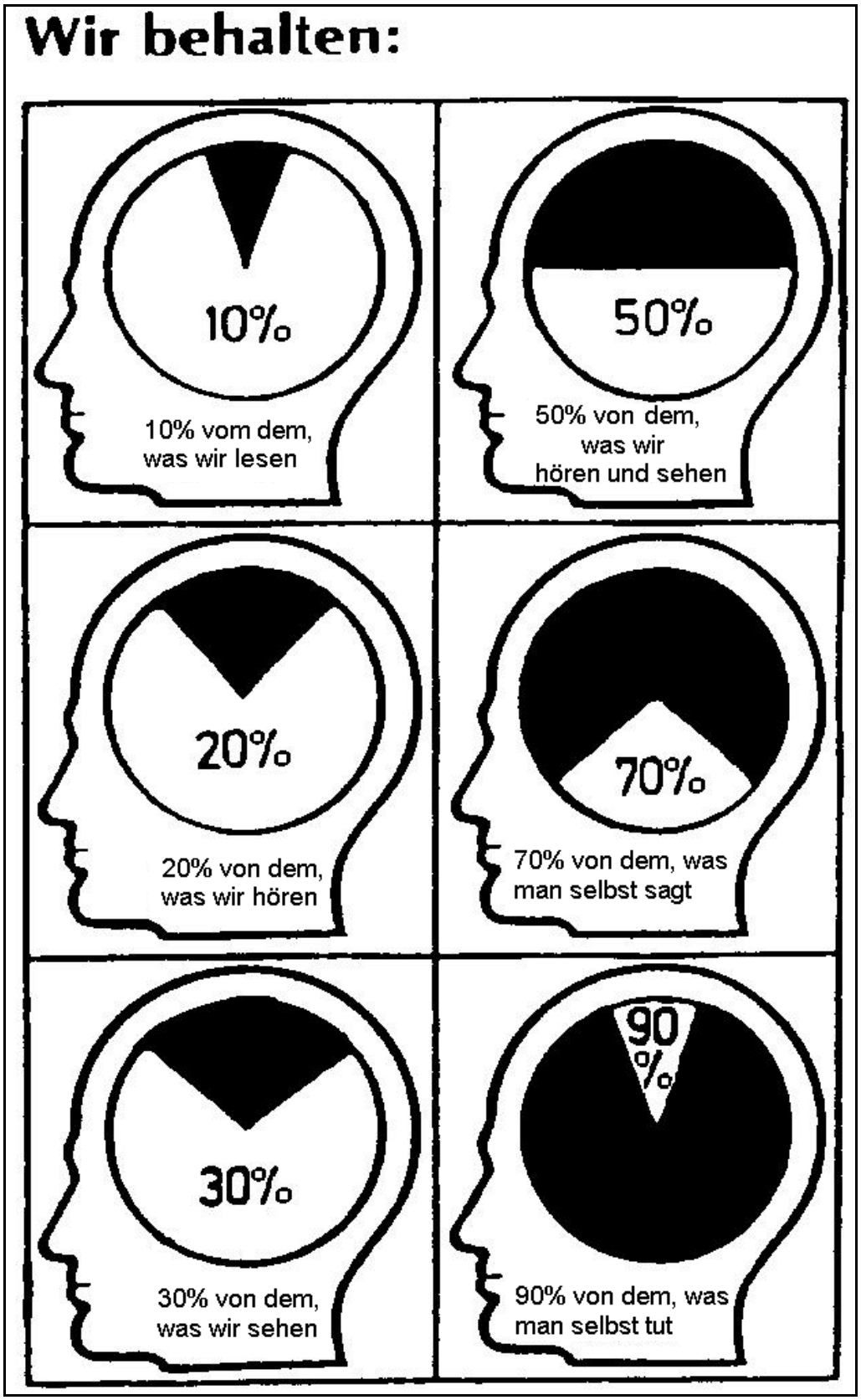


Abbildung 1: Analyse des Battelle-Instituts[15]

[15] Battelle-Institut eV; Am Römerhof 35; 60486 Frankfurt

Das lernpsychologische Modell fußt auf der These, dass der handelnde Mensch in besonderer Weise in der Lage ist zu lernen. Diese These wird zusätzlich von einer Analyse des Battelle-Institutes (Abbildung 1) gestützt. Danach behalten wir 90% von dem, was wir selbst tun. Wir können aber nur 10% von dem behalten, was wir ausschließlich lesen.

Das Studieren an der FernUniversität, auch oft noch an der virtuellen FernUniversität, ist im wesentlichen auf das Lesen der Kursunterlagen und das Lösen der Einsendeaufgaben beschränkt. Hier kann das Programm einen weiteren Eingangskanal anbieten und dadurch folgendes ermöglichen:

- Experimentelles Vorgehen
- Entdeckendes Lernen[16]
- Üben und Verfestigen des betreffenden Kurstextteils von [FU1840]
- Lösen von Einsendeaufgaben

Mit dem Programm wird zusätzlich bei den Benutzern der visuelle Eingangskanal[17] angesprochen. Dies erscheint uns besonders wichtig, denn:

- „Je mehr Arten der Erklärung angeboten werden, je mehr Kanäle der Wahrnehmung genutzt werden, ... desto fester wird Wissen gespeichert, desto vielfältiger wird es verankert und auch verstanden, desto mehr Schüler werden den Wissensstoff begreifen und ihn später auch erinnern.“ [VEST78, Seite 42]

- „Unterricht besteht darin, dem Schüler Schritt für Schritt durch Formulierungen und Neuformulierungen eines Problems oder Wissensbereiches einen Weg zu weisen.” [STAM84, S. 63]

[16] Die mit dem entdeckenden Lernen vertretene Hauptthese lautet, dass das Lernen um so wirkungsvoller ist, als es durch eigene aktive Erfahrungen erfolgt und dabei auf selbständigen entdeckerischen Tätigkeiten beruht.

[17] Vester [VEST78] unterscheidet zwischen vier Eingangskanälen:

auditiv	Das Verstehen erfolgt durch Kommunikation.
visuell	Das Lernen erfolgt durch das Auge, durch Beobachtung.
haptisch	Das Erfahren erfolgt durch Fühlen und Anfassen.
intellekt	Das Lernen erfolgt anhand abstrakter Formeln.

Auch wenn sich diese Zitate auf Schule beziehen, lassen sie sich nach unserer Erfahrung als Studenten und Referenten in der Erwachsenenbildung[18] sinngemäß auch auf Studenten der Informatik übertragen.

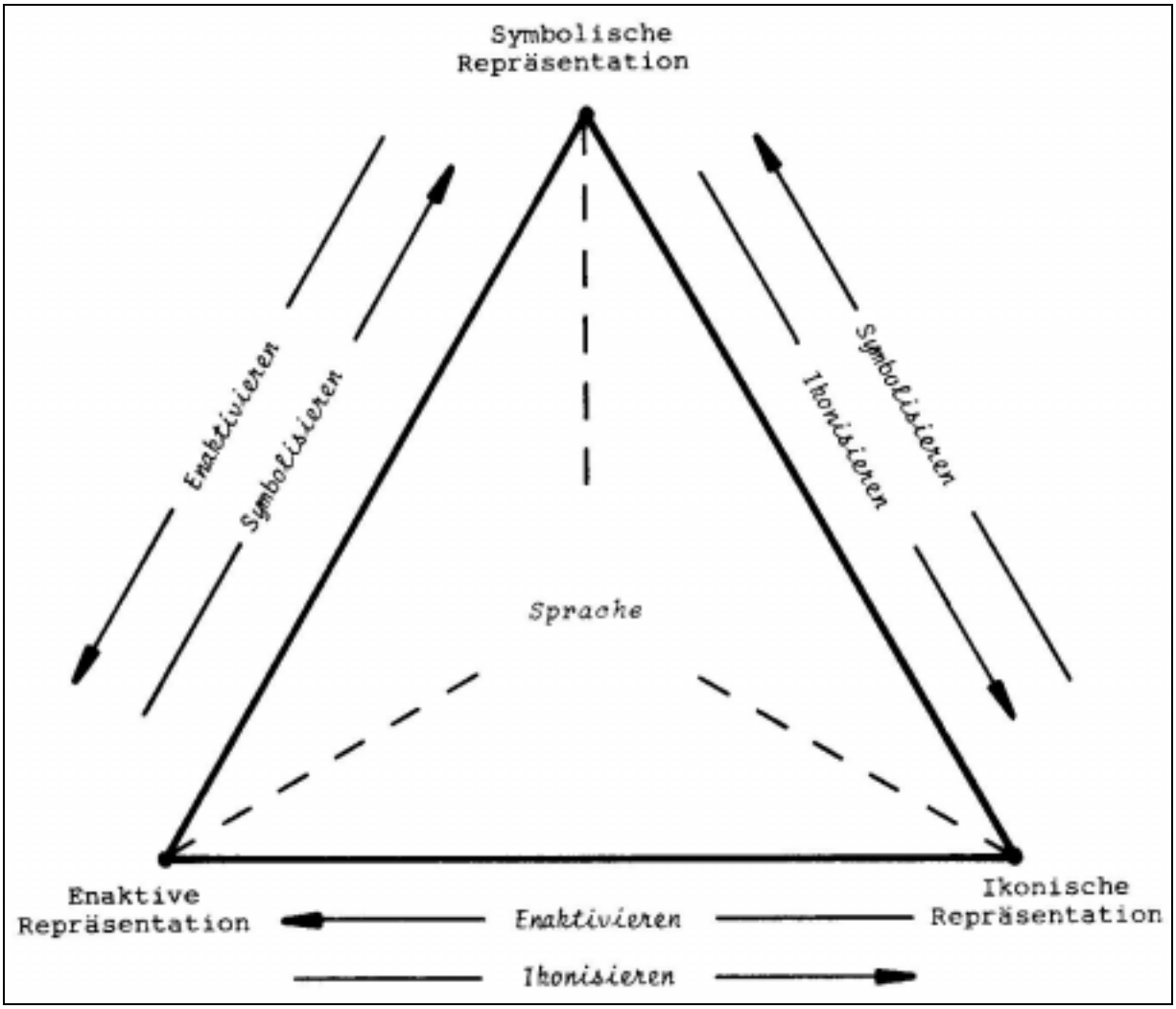


Abbildung 2: Repräsentationen nach Bruner[19]

Beim Bearbeiten und Erfassen des 2-d-Baums und des Prioritätssuchbaums aus dem Kurs Algorithmische Geometrie [FU1840, Seiten 118 bis 125 und 130 bis 135] werden die Benutzer besonders bezüglich des Begriffslernens und des Problemlösens gefordert. Beim Begriffslernen

[18] Wir sind beide seit Jahren im Bereich der Erwachsenenbildung tätig und führen Aus- und Fortbildungsveranstaltungen durch.

[19] Aus [STAM84, Seite 64]

können u. a. an Einzelbeispielen erfahrene Wahrnehmungen zusammengefasst werden. Dies bedeutet, die zu erarbeitenden neuen Begriffe in die vorhandene kognitive Struktur einordnen zu können. Durch den Einsatz des Variationsprinzips, also die Darstellung der Beispiele in verschiedenen Darstellungsformen[20], ist eine weitere Verstärkung erzielbar. Dieser lerntheoretische Ansatz wird auch durch das Modell von Bruner (Abbildung 2) gestützt.

Nach diesem Modell (Abbildung 2) ist der Wissensbereich „Suchbäume" auf drei verschiedene Arten darstellbar:

- **Enaktiv**
  Das Lehrziel soll durch eine entsprechende Anzahl von Handlungen erreicht werden.

- **Ikonisch**
  Durch eine Reihe zusammenfassender Bilder oder Grafiken wird eine bestimmte Konzeption versinnbildlicht. Dabei erfolgt allerdings keine vollständige Definition der Begriffe.

- **symbolisch**
  Dies wird repräsentiert durch eine Folge von Lehrsätzen und Definitionen.

Basierend auf einer kognitiven Psychologie möchten wir mit Bruner feststellen:

Der Lernprozess lässt sich wesentlich verbessern, wenn der Wechsel zwischen den drei Ebenen von den Benutzern bewusst vollzogen wird.

Eine weitere Position von Bruner ist, besonders das entdeckende Lernen zu fördern. Mit dieser Aussage verdeutlicht er seine starke Nähe zum von Freudenthal vertretenen genetischen Prinzip.

[20] Dies sind hier der Kurstext und die Visualisierung durch unser Programm.

Damit ergibt sich eine interessante Alternative zu einer an der systematisch geordneten Lehrbuchstruktur orientierten Lehr-Lern-Organisation, wie sie bei den Buch- und Online-Versionen der Informatikkurse an der FernUniversität existiert.

Mit dem Kurstext liegt eine symbolische Repräsentation vor. Lernende sind aber häufig enaktiv[21] veranlagt. Mit unserem Softwareprodukt kann durch gezielten Einsatz erreicht werden, dass durch eine umfangreiche Anzahl von Handlungen die zu erlernenden Begriffe schrittweise auf den drei unterschiedlichen Repräsentationsebenen erschließbar werden. Diese Variation bedingt aber eine gezielte Einbindung des Programms in die Kurseinheit 3 von [FU1840] und/oder die Einbindung von Aussagen der Kurseinheit 3 in das Hilfesystem des Softwareproduktes.

Welche Möglichkeiten bieten sich nun für den adressatengerechten und effizienten Einsatz unseres Softwareproduktes im Rahmen des Kurses „Algorithmische Geometrie" an der FernUniversität Hagen?

Schlussfolgernd sehen wir folgende Einsatzmöglichkeiten für das von uns zu erstellende Programm:

- Der Benutzer erlebt experimentell oder per "Trial and Error" eine erste enaktive Einführung in den 2-d-Baum und den Prioritätssuchbaum.

- Der Benutzer festigt oder vertieft durch den Einsatz des Programms seine Kenntnisse über den 2-d-Baum und den Prioritätssuchbaum.

- Das Lehrgebiet Praktische Informatik VI nutzt das Programm bei der Erstellung von Einsendeaufgaben.

- Der Benutzer erlebt mit den Demonstrationen eine Visualisierung der Zusammenhänge beim 2-d-Baum und Prioritätssuchbaum.

[21] Nach Abbildung 1 übernehmen wir 90% von dem, was wir selbst tun, in unser Langzeitgedächtnis.

Diese verschiedenen Einsatzmöglichkeiten müssen nun von uns bei der Planung der Software berücksichtigt werden.
Hieraus ergeben sich unabhängig von der noch durchzuführenden Elaboration-Phase[22] folgende bei der Softwareentwicklung zu berücksichtigende Aspekte:

- Die Bilder des Eingabe- und des Ausgabebereiches sollen für die weitere Bearbeitung mit einer Textverarbeitung oder einem Bildbearbeitungsprogramm in einem Standardgrafikformat wie TIFF, JPEG oder GIF speicherbar sein.

- Eingabe- und Ausgabebereiche lassen sich auf einem Drucker ausgeben.

- Dem Benutzer werden animierte Demonstrationen zur Veranschaulichung des Aufbaus von 2-d-Baum und Prioritätssuchbaum zur Verfügung gestellt.

- Der Benutzer kann durch animiertes oder schrittweises Eintragen der Punkte in den 2-d-Baum oder den Prioritätssuchbaum erfahren, wie die in [FU1840, Seiten 118 bis 125 und 130 bis 135] beschriebenen Algorithmen arbeiten.

- Durch die Möglichkeit, auch mit den entarteten Suchbäumen[23] arbeiten zu können, wird dem Benutzer in Ansätzen ein Einblick in ihr Laufzeitverhalten ermöglicht. Gleichzeitig erfährt er anschaulich, worin der Vorteil der in [FU1840, Seiten 118 bis 125 und 130 bis 135] beschriebenen ausgeglichenen Datenstrukturen besteht.

[22] Siehe auch Kapitel 7
[23] Siehe Ausführungen zum dynamischer 2-d-Baum in den Kapiteln 7.4.3.1 und 7.4.3.2

5. Softwareergonomische Aspekte

In diesem Kapitel wollen wir im Rahmen der Inception-Phase[24] Grundsätze für den Aufbau der Benutzeroberfläche und die Programminteraktionen entwickeln. Hierzu stellen wir uns folgende Frage:

Welche Aspekte der Software-Ergonomie sind im Rahmen der Entwicklung des Programms `VisualGeom`[25] wichtig?

Dazu müssen wir zunächst klären, was unter Software-Ergonomie zu verstehen ist und welche Intentionen mit der Software-Ergonomie vertreten werden. Hierzu findet sich in [DUDE88, Seite 218]:

> „Die Software-Ergonomie entwickelt Kriterien und Methoden zur Gestaltung interaktiver Programmsysteme, die den menschlichen Bedürfnissen der Benutzer nach einer ausgeglichenen Verteilung der geistigen, körperlichen und sozialen Belastung weitgehend entgegenkommt."

Das Ziel der Software-Ergonomie besteht danach darin, eine effiziente Ausführung der beschriebenen Aufgabe zu gewährleisten. Dabei müssen mögliche negative physische und psychische Auswirkungen minimiert werden. Vielmehr sollen Kompetenzen beim Umgang mit dem Softwareprodukt entwickelt werden, der Benutzer seine Persönlichkeit durch den Umgang mit den Softwareprodukt weiterentwickeln können. Dies sollte unseres Erachtens nach die wesentliche Intention einer Lehr-Lern-Software darstellen.

[24] Siehe auch Kapitel 6

[25] Da die Aufgabe unserer Diplomarbeit ist, **geom**etrische Datenstrukturen zu **visual**isieren, haben wir das Programm **VisualGeom** genannt. Wir werden nachfolgend grundsätzlich diesen Namen in unseren Ausführungen zu verwenden.

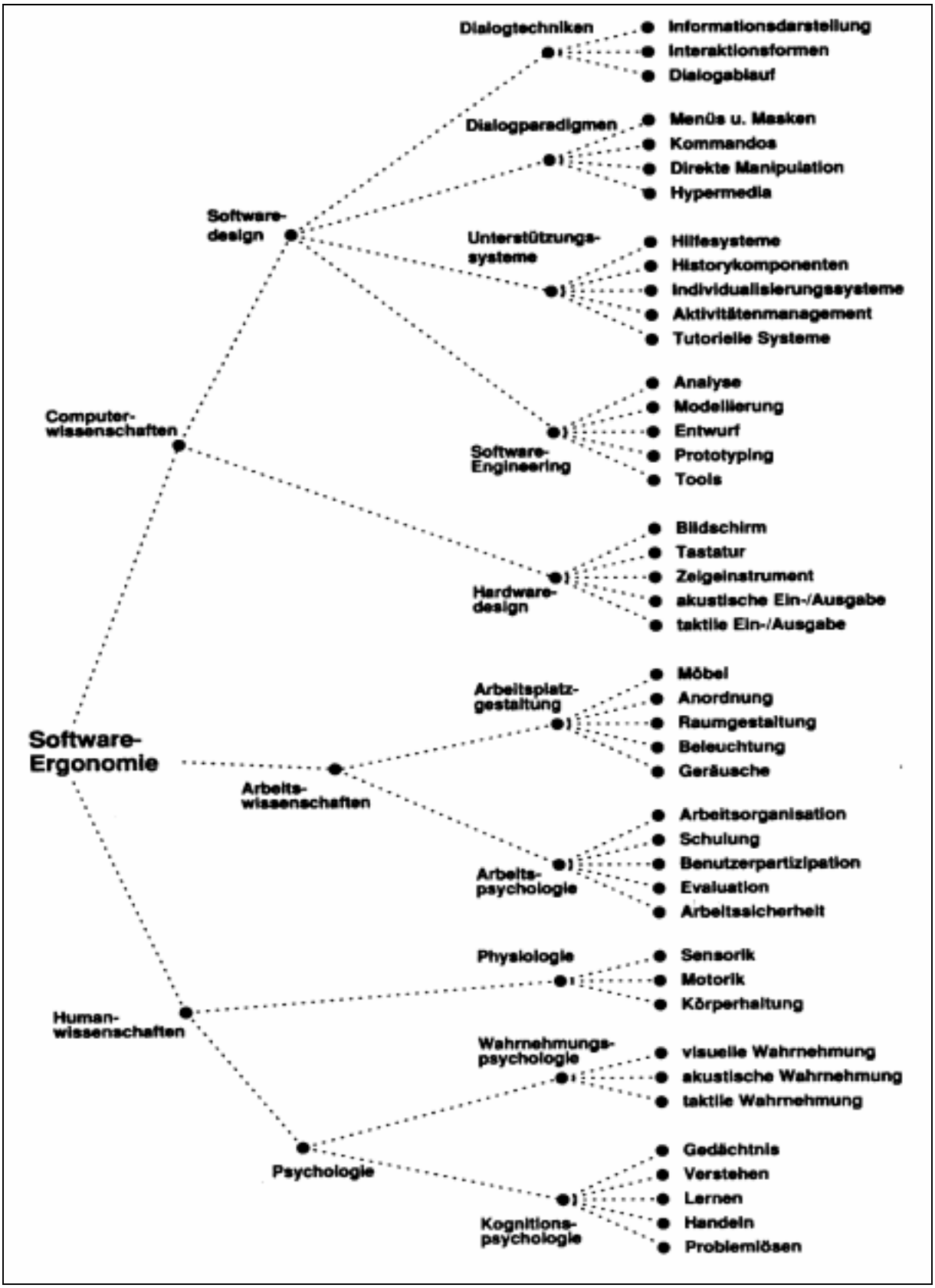


Abbildung 3: Der Software-Ergonomie zuzuordnende Themengebiete[26]

[26] [HERC94, Seite 7]

Für unsere Softwareentwicklung erscheint uns die Definition aus [DUDE88] zu allgemein gehalten. Um Antworten auf unsere Frage von Seite 17 zu erhalten, wenden wir uns deshalb der Abbildung 3 zu. Diese Klassifikation zeigt einen Überblick über alle Themen, die der Software-Ergonomie zugeordnet werden können.

Eine ausführliche Auseinandersetzung mit der Software-Ergonomie würde den Rahmen unserer Diplomarbeit sprengen. Wir verweisen deshalb auf die im Literaturverzeichnis genannten Bücher[27]. Wir greifen hier nur die unseres Erachtens für unsere Softwareentwicklung besonders zu berücksichtigenden Aspekte auf.

Als Konsequenz für die Inception-Phase[28] im Rahmen der Diplomarbeit sahen wir es als erforderlich an, grundsätzlich die Aspekte

- Dialogtechniken und Dialogparadigmen,
- Unterstützungssysteme,
- Physiologie,
- Wahrnehmungspsychologie und
- Kognitionspsychologie

zu berücksichtigen.

Hieraus haben wir die nachfolgend dargestellten Schlussfolgerungen für die Entwicklung von **`VisualGeom`** gezogen.

Schlussfolgerungen für `VisualGeom` zu Dialogtechniken und Dialogparadigmen

- Die Informationsmenge pro Bildschirmseite werden wir auf die Punkte und die Graphen beschränken. Hierdurch wird der Umfang der Informationsmenge übersichtlich gehalten. Fehlermeldungen werden in der Statuszeile ausgegeben. Geeignet gestaltete Hervorhebungen bei Animationen und Anfragen sollen für den Benutzer verständnisfördernd wirken.

[27] siehe hierzu auch [HETT80], [KOCH91], [HAM92], [WAND93], [HERC94], [OPPE94]
[28] siehe auch Kapitel 6

- Die Darstellung von 2-d-Baum und Prioritätssuchbaum soll beim Einfügen, Löschen und bei der Anfrage durch farbliche Hervorhebung oder dynamischen Aufbau visualisiert werden. Dabei wird der geeignete Einsatz der Farben unter Berücksichtigung der aus der Literatur bekannten Farbcodierungen[29] erfolgen.
- Für 2-d-Baum und Prioritätssuchbaum wollen wir die Oberfläche, die Fenster und das Layout einheitlich gestaltet. Dabei werden wir Menüs als selektionsorientierte Interaktionsformen auswählen.

  Auf den Einsatz einer Symbolleiste für die unmittelbare Auswahl bestimmter Menüpunkte wird aus Zeitgründen verzichtet. Dies gilt ebenfalls für Hotkeys oder kontextbezogene Pop-up-Menüs. Dies wären sinnvolle Erweiterungen einer späteren Release von **`VisualGeom`**.

  Bei der Anordnung der Menüelemente und der Reihenfolge der Menüpunkte erscheint es uns besonders wichtig, bei den Benutzern Erwartungskonformität zu erreichen. Dies bedeutet, die Sortierung der Menüpunkte grundsätzlich nach bei anderen Softwareprodukten[30] bekannten Kriterien vorzunehmen. Dabei werden die Auswahlmenüs zustandsabhängig sein, also Menüpunkte abhängig von der jeweils ausgewählten Strukturvariante[31] aktiviert oder deaktiviert werden. Hierdurch erlebt der Benutzer, welche Möglichkeiten der jeweilige Modus ihm anbietet.
- Wir wollen bei der Softwareentwicklung die Anforderungen an die Dialoggestaltung nach ISO9241[32] berücksichtigen. Hierzu gehören u. a.:
  - Aufgabenangemessenheit wollen wir mit einem an die jeweilige Aufgabe angepassten Dialog erzielen.
  - Die Selbstbeschreibungsfähigkeit wollen wir unter anderem mit der Aufteilung der Zeichenfläche in den Eingabebereich und den Ausgabebereich erreichen.
  - Die Steuerbarkeit soll durch einstellbare Geschwindigkeit beim Aufbau von 2-d-Baum und Prioritätssuchbaum erreicht werden. Hierzu werden wir im Menü für die „Art des Zeichnens“ abhängig von der jeweiligen Strukturvariante die Auswahl zwischen „unmittelbar“, „animiert“ und „schrittweise“ anbieten.

[29] Siehe auch [HERC94, Seiten 73 bis 77]

[30] Hierbei denken wir an Textverarbeitungsprogramme wie WinWord von Microsoft oder das Officepaket der Firma Star Division.

[31] Hiermit meinen wir die geplante Auswahl beim 2-d-Baum zwischen „dynamisch“, „dynamisch interaktiv“, „ausgewogen“ und „allgemein“ und beim „Prioritätssuchbaum zwischen „semidynamisch statisch“, semidynamisch“, „dynamisch“ und „dynamisch balanciert“. Siehe auch Kapitel 7, 8 und 10.

[32] Siehe hierzu auch [FU4763, S. 89]

- Der Dialog beim interaktiven dynamischen 2-d-Baum soll so aufgebaut werden, dass zunächst die Splitkoordinate und erst dann der Splitwert abgefragt werden. Damit wollen wir beim Benutzer das Gefühl der Konsistenz des Systems und Berechenbarkeit des Systemverhaltens erreichen und damit möglichst ihrer Erwartungskonformität entsprechen.
- Um Übersichtlichkeit zu gewährleisten, muss eine geeignete Darstellung und Anordnung der Informationen auf dem Bildschirm vorgenommen werden. Hierzu gehören vor allem Orts- und Farbcodierung[33]. Dies wollen wir u.a. erreichen, indem wir das Einfügen und Löschen und das Erstellen von Anfragen in geeigneter Weise farblich hervorheben.

Schlussfolgerungen für `VisualGeom` zu Unterstützungssystemen

- Ein Online-Hilfesystem ist in **`VisualGeom`** einzubinden. Dabei wollen wir folgende Ziele verfolgen:
  - Erklärung des Anwendungsbereiches des Programms und der Grenzen des Systems
  - Erklärung der konkreten Anwendungsziele
  - Erklärung der Objekte und Operatoren des Anwendungsbereiches
  - Erklärung der syntaktischen Eingaberegeln und Korrekturvorgänge
  - Belegung der Tasten des Zeigeinstrumentes
  - Demonstrationen der Möglichkeiten unseres Programms
- Im Hilfesystem sollen Querverweise zum Kurstext [FU1840] enthalten sein.
- Uns erscheint sinnvoll, im Text und der Onlineversion von [FU1840] einen Querverweis auf **`VisualGeom`** vorzusehen.
- Als Ergänzung wird auch eine Bedienungsanleitung in schriftlicher Form[34] angeboten.[35]

Schlussfolgerungen für `VisualGeom` zur Physiologie

Die Eingabe der Punkte soll mit der Maus erfolgen. Bei der Auswahl der Punkte im Eingabebereich werden wir nur Punkte mit ganzzahligen Koordinaten verwenden. Dies wird erreicht durch Fang und Raster für das Eingabesignal von linker oder rechter Maustaste.

[33] Siehe auch [HERC94, Seiten 73 bis 77]
[34] Siehe auch Kapitel 10
[35] Diese Anleitung ist als eigenständige Postscript-Datei Bestandteil von **`VisualGeom`**.

Hierdurch hoffen wir, Zeigefehler durch physiologisches Rauschen[36] minimieren zu können.

Aus gleichem Grunde wollen wir beim interaktiven dynamischen 2-d-Baum auf eine Mauseingabe der Splitgeraden verzichten. Bereits bei einem Rasterabstand von 10 Bildpunkten erwies sich das Platzieren einer Splitgeraden als schwierig. Bei einem Rasterabstand von 2 Bildpunkten wurde es unmöglich[37]. Deshalb wird nur die gezielte Eingabe per Abfrage zugelassen.

Schlussfolgerungen für `VisualGeom` zur Wahrnehmungspsychologie

- Wir wollen eine möglichst erwartungskonforme Gestaltung der Menüs vornehmen.
- Der Einsatz von Farben soll unter Berücksichtigung der Farbempfindlichkeit des menschlichen Auges [HERC94, S.56,57] und der psychologischen Wirkungen von Farben erreicht werden. Dies wollen wir bei
  - der Visualisierung von Zuständen und Zustandsänderungen,
  - dem Unterscheiden von Zuständen,
  - dem Lenken der Aufmerksamkeit auf wichtige Bildschirminhalte,
  - dem Markieren selektierter Bildschirmobjekte und
  - dem Herstellen von Informationsbeziehungen

  berücksichtigen.

  Farben sollen gezielt und sparsam verwendet und mit fester Bedeutung durchgängig für den 2-d-Baum und den Prioritätssuchbaum benutzt werden. Besonders denken wir dabei an den Einsatz der Farbe „rot", mit der sich u. a. Nähe und Aufmerksamkeit[38] erzeugen lässt.
- Zeichen und Hintergrundfarbe sollen ergonomisch richtig kombiniert werden. Dabei ist auch auf die richtige Gestaltung von Zeichengröße, Gestalt der Zeichen, Zeilenabstand und Zeichenkontrast zu achten.
- Um akzeptable Antwortzeiten beim Einsatz unseres Java-Programms **`VisualGeom`** zu erhalten, sehen wir uns gezwungen, Minimalanforderungen an die Hardwarekonfiguration zu formulieren.

[36] Hiermit sind nach [HERC94, Seite 69] Zeigefehler gemeint, die vor allem durch Muskelzittern verursacht werden und ein punktgenaues Treffen verhindern.

[37] Dies konnte auf einem Laptop mit einem 13,3" TFT-Display und einer Auflösung von 1024 * 768 Bildpunkten festgestellt werden.

[38] Siehe auch [HERC94, Seiten 73 bis 77]

Schlussfolgerungen für `VisualGeom` zur Kognitionspsychologie

- Der Benutzer soll die Aufgabenstellung durchschauen können. Es soll für ihn erkennbar sein, welche Auswirkungen sein Eingreifen beim Einfügen oder Löschen eines Punktes auf den 2-d-Baum oder den Prioritätssuchbaum oder beim Erzeugen von Anfragen besitzt.
- **`VisualGeom`** soll dem Benutzer stets aktuelle Informationen zum ausgewählten Baumtyp anzeigen. Bei Fehleingaben sind verständliche und konstruktive Meldungen in der Statuszeile auszugeben.
- Besonderes Augenmerk wollen wir auf die Fehlerrobustheit des Systems und die einheitliche Strukturierung der Fehlermeldungen legen.
- Der Benutzer soll einen angemessenen Entscheidungsspielraum erhalten. So wird ihm beim 2-d-Baum und Prioritätssuchbaum die Auswahl zwischen dynamischem und statischem Aufbau und die Auswahl zwischen den Arten unmittelbar, animiert und schrittweise ermöglicht.

6. Objektorientierte Softwareentwicklung

Aus den Ausführungen im Kapitel 3 Rahmenbedingungen für die Softwareentwicklung folgt, das Programm in einer objektorientierten Sprache zu implementieren. Als logische Konsequenz ergibt sich, auch für die Softwareentwicklung ein geeignetes objektorientiertes Vorgehensmodell einzusetzen. Hierfür bieten sich verschiedene Modelle an.
Abbildung 4 vermittelt einen Überblick über die Entwicklung der verschiedenen objektorientierten Vorgehensmodelle. Danach kann der Softwareentwickler aktuell für seine objektorientierte Softwareentwicklung zwischen sieben verschiedenen Vorgehensmodellen auswählen.

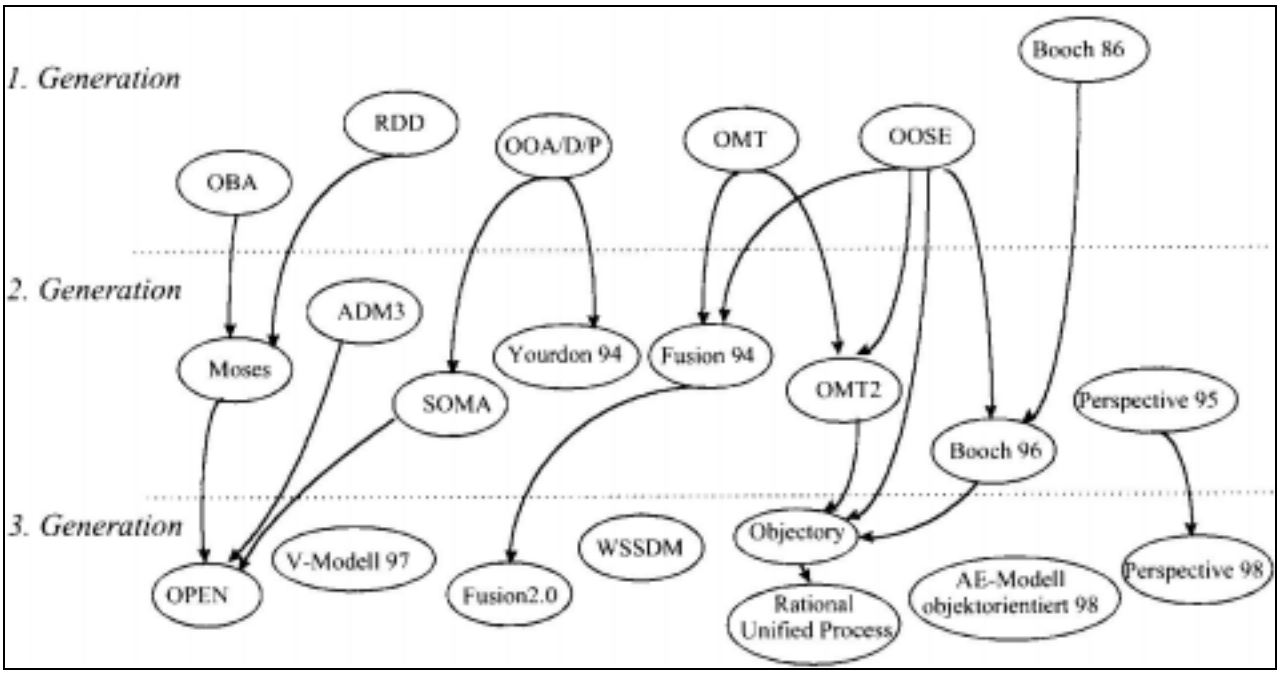


Abbildung 4: Entwicklung objektorientierter Vorgehensmodelle[39]

Aufgrund der in der Praxis inzwischen recht guten Akzeptanz des **R**ational **U**nified **P**rocess [BOOC99], der auch als **RUP** bezeichnet wird, und der vorhandenen Unterstützungswerkzeuge von der Firma Rational[40], haben wir beschlossen, unsere Softwareentwicklung am RUP anzulehnen.

[39] Siehe auch [IS9922, S. 167]
[40] Rational Software Corporation, 2800 San Tomas Expy, Santa Clara ca 95051-9813

Was sind die wesentlichen Eigenschaften des Rational Unified Process?

Schwerpunkte des Rational Unified Process sind:

- Die Orientierung an Anwendungsfällen (Use Cases)
- Die Architekturzentrierung
- Die iterative und inkrementelle Vorgehensweise

Mit dem Rational Unified Process von Abbildung 5 erheben die drei Autoren[41] den Anspruch, den gesamten Lebenszyklus eines Softwareproduktes begleiten zu können.

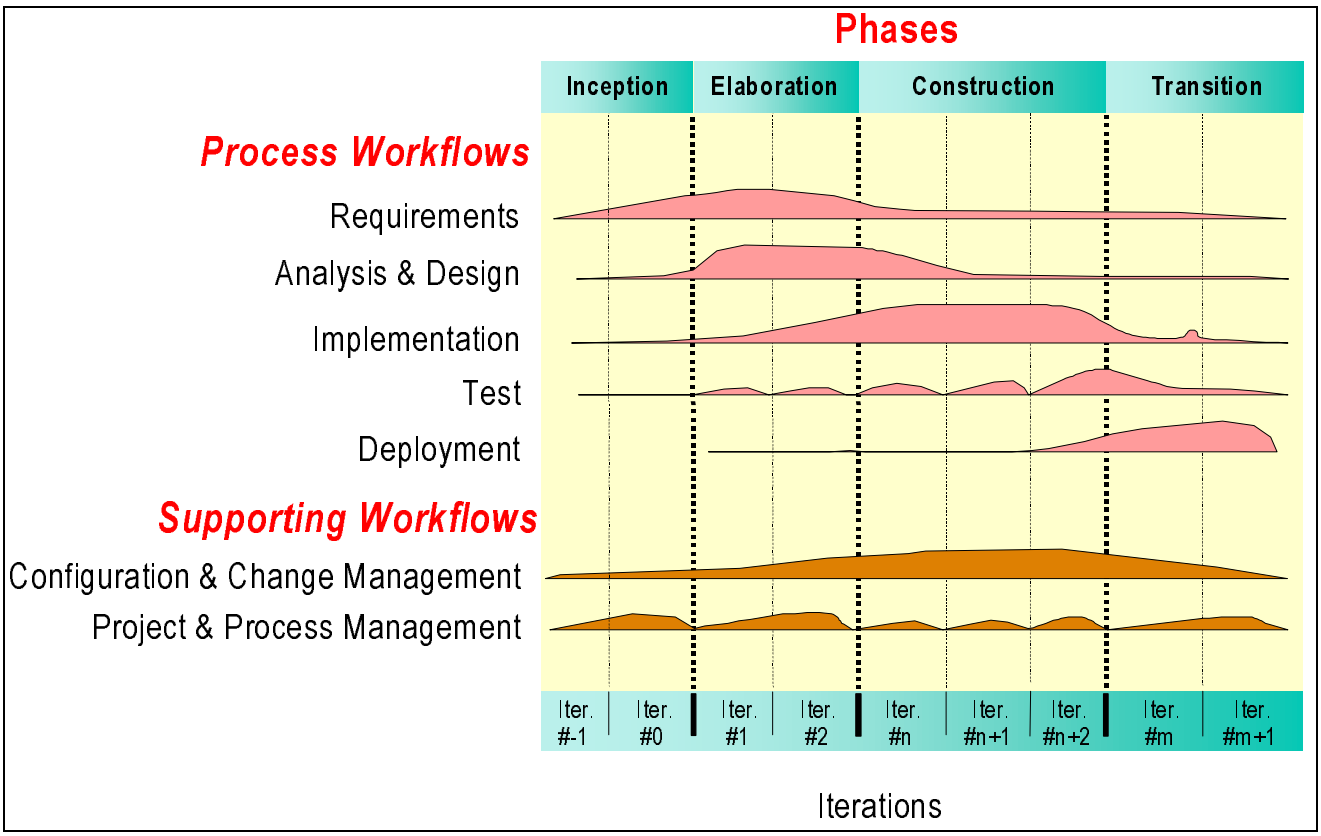


Abbildung 5: Der Rational Unified Process[42]

[41] RUP wurde von den drei „Amigos“ Booch, Jacobson und Rumbaugh entwickelt. Sie sind ebenfalls an der Entwicklung von UML [BOOC99a] beteiligt und inzwischen alle bei der Firma Rational beschäftigt.

[42] Diese Abbildung wurde von uns nach [BOOC99, S. 11] erstellt.

Aus der Abbildung 5 wird ersichtlich, dass beim RUP zwischen Phasen (Phases) und Arbeitsabläufen (Workflows) unterschieden wird. Sie teilen den Gesamtprozess zeitlich horizontal und vertikal auf. Jeder Phase sind die verschiedenen Arbeitsabläufe mit unterschiedlicher Intensität, erkennbar an der Höhe der einzelnen Graphen, zugeordnet.

Im Rational Unified Process werden die Phasen unterteilt in:

- **Inception** / Voruntersuchung

  Es werden schwerpunktmäßig die Arbeitsabläufe aus der Geschäftsmodellierung und der Anforderungsanalyse (Requirement gemäß Abbildung 5) durchgeführt. Dabei sind nach [BOOC99, S. 12] die entscheidenden Fragen in der Inception-Phase:

  What is the system primarily going to do for each of its major users?
  What could an architecture for that system look like?
  What is the plan and what will it cost to develop the product?

  Die für die Entwicklung von **`VisualGeom`** erforderlichen Elemente der Inception-Phase haben wir bereits in den Kapiteln 2, 3, 4 und 5 durchgeführt. Dort wurden die Eingangsvoraussetzungen der Benutzer beschrieben, die Rahmenbedingungen für die Softwareentwicklung und Grundelemente des äußeren Aufbaus von **`VisualGeom`** entworfen. Aussagen über die Kosten der Softwareentwicklung sind nur bezogen auf den zeitlichen Rahmen möglich, der sich aus der Bearbeitungsdauer für die Diplomarbeit von insgesamt sechs Monaten ergibt.

- **Elaboration** / Systemanalyse und Entwurf

  In dieser Phase erfolgt u. a. die Überprüfung der Machbarkeit, die Formulierung visionärer Anforderungen und die Festlegung der verwendeten Architekturmuster bzw. des Architekturstils. Dabei werden die wichtigsten Anwendungsfälle im Detail spezifiziert und die Systemarchitektur entworfen. Das Ergebnis ist die Architektur als Sicht aller Modelle des Systems (Abbildung 6). Mit dem Ende dieser Phase sollen nach den Autoren die kritischen Anwendungsfälle und die grundlegende Architektur realisiert sein.

Die entscheidende Frage für die Elaboration-Phase stellt sich laut [BOOC99, S. 12] mit:

> **Are the use cases, architecture, and plans stable enough, and are the risks under sufficent control to be able to commit to the whole development work in a contract?**

Das Kapitel 7 enthält Ausführungen zu dieser Phase bei der Entwicklung unseres Programms **`VisualGeom`**.

- **Construction** / Erzeugung

  Das Softwareprodukt wird erstellt, indem die Anwendungsprogrammierer alle Anwendungsfälle nacheinander implementieren. Am Ende dieser Phase ist das Softwareprodukt zwar vollständig, befindet sich aber ggf. noch in einem Alpha-Stadium. Entscheidend für diese Phase ist nach [BOOC99, S. 12] die Frage:

  > **Does the product meet users' needs sufficiently for some customers to take early delivery?**

  Die Kapitel 8 und 9 enthalten Ausführungen zu dieser Phase bei der Entwicklung unseres Programms **`VisualGeom`**.

- **Transition** / Einführung

  In dieser Phase liegt der Schwerpunkt auf dem Testen des Programms unter Realweltbedingungen, der Beseitigung von Fehlern und dem Durchführen noch erforderlicher Programmmodifikationen. Ziel sollte dabei die Herstellung der Qualitätssicherung nach ISO9001 sein. Des weiteren erfolgt die Schulung der Benutzer und die Auslieferung des Softwareproduktes.

  Diese Phase kann für **`VisualGeom`** erst nach dem Ende der Diplomarbeit erfolgen.

Im Rahmen der Softwareentwicklung mit dem Rational Unified Process werden in jeder der vier Phasen pro Iteration die einzelnen Arbeitsschritte / **Process Workflows** - Requirement, Analysis & Design, Implementation, Test - nacheinander durchlaufen. Je nachdem, in welcher aktuellen Phase sich das Produkt gerade befindet, ergibt sich nach Abbildung 5, mit welcher Intention der betreffende **Process Workflow** bearbeitet wird.
In den frühen Iterationen werden die Anforderungen, die Probleme und Risiken konkretisiert. In den späteren Iterationen kommt jeweils ein weiterer Baustein zum Programm hinzu. Es wächst inkrementell.
Deshalb bezeichnen Booch, Jacobson und Rumbaugh den Prozess der Softwareentwicklung mit dem Rational Unified Process auch als iterativ und inkrementell.

Zur Veranschaulichung der in der **Elaboration**-Phase entwickelten Architektursicht des Systems verwenden Booch, Jacobson und Rumbaugh die 4+1-Sichtendarstellung von Abbildung 6:

- **Use Case View** / Anwendungsfallsicht
  Die Anwendungsfallsicht beschreibt das funktionale Verhalten des Softwareproduktes aus der Sicht von Benutzern und Entwicklern. Es werden z. B. mittels **UML**, der **U**nified **M**odeling **L**anguage, die Interaktionen zwischen den Benutzern und dem Anwendungssystem bei der Durchführung eines Arbeitsganges beschrieben.

  Hierzu stellt [BOOC99] fest:

> **What do you want the system to do for each user?**[43]
> **A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor.**[44]

[43] [BOOC99, Seite 38]
[44] [BOOC99, Seite 41]

- **Design View** / Entwurfssicht

  Mit der Beschreibung der Objekte, Schnittstellen und Beziehungen wird die statische Struktur des Systems als Menge von Subsystemen, Klassen und Interfaces festgelegt. Die Anwendungsfälle werden dabei mittels **UML** zusätzlich in Kollaborationsdiagrammen dargestellt.

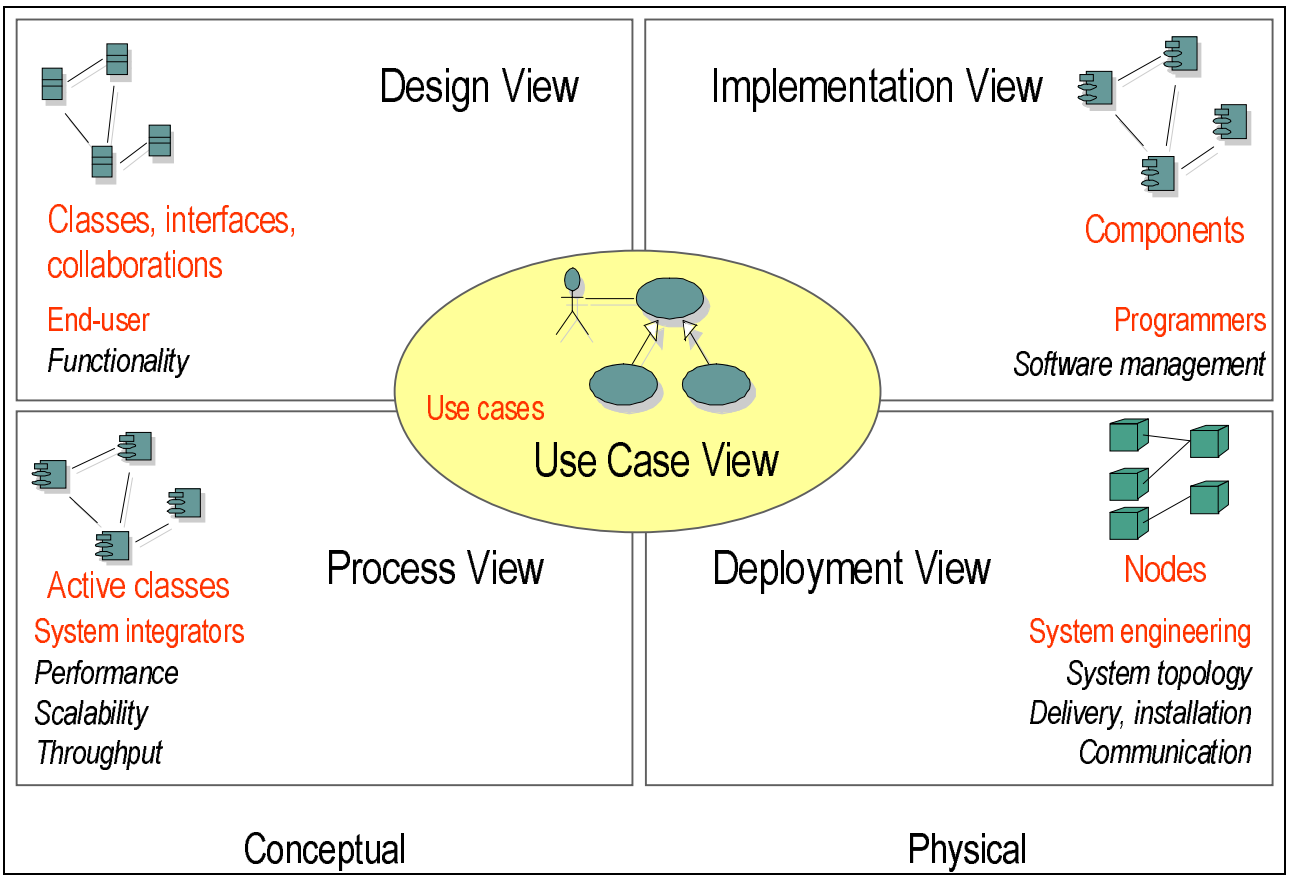


Abbildung 6: 4+1-Sichten[45]

- **Process View** / Prozesssicht

  Die Prozesssicht beschreibt die Nebenläufigkeit und Synchronisation für die aktiven Klassen. Es erfolgt die Visualisierung der Anwendungsfälle mittels Aktivitätsdiagrammen. Des weiteren werden Objekte und ihre Beziehungen einschließlich der zeitlichen Ordnung ihres Nachrichtenaustausches mit Sequenzdiagrammen veranschaulicht.

[45] Dieses Abbildung entstand nach Folien der Firma Rational-Software vom Rational Worldwide Software Symposium in 1999.

- **Implementation View** / Implementierungssicht

  Sie enthält die Komponenten mit ihren Klassen und Paketen einschließlich der Beziehungen zwischen den Komponenten.

- **Deployment View** / Einsatzsicht

  Die Einsatzsicht wird mittels Verteilungsdiagrammen veranschaulicht. Diese verdeutlichen, welche Komponenten und Objekte auf welchen Prozessen oder Computern lauffähig sind. Dabei wird beschrieben, wie diese konfiguriert sind und welche Kommunikationsbeziehungen zwischen ihnen bestehen.

Bei der Anwendung des Modells RUP ergibt sich für uns als Konsequenz, zunächst im Rahmen des Requirements alle Anwendungsfälle zu ermitteln, Analyse- und Designentscheidungen zu treffen und dann die Anwendungsfälle iterativ und inkrementell zu implementieren.

7 Elaboration / Systemanalyse und Entwurf

Für unser kleines Programm lehnen wir uns nun an das in Kapitel 6 beschriebene Prinzip an, indem wir anwendungsfallorientiert vorgehen und zur Dokumentation Diagramme in UML-Notation[46] verwenden. Die Anwendungsfälle dienen zur Ermittlung der Anforderungen und zur groben Abstimmung des zukünftigen Systemverhaltens bzgl. der grafischen Benutzeroberfläche.
Die detaillierteren Ausführungen zu den Anwendungsfällen setzen wir in Aktivitätsdiagramme um und entwickeln daraus zusammen mit den Detailbeschreibungen zu den Anwendungsfällen erste grobe Klasseneinteilungen.

7.1 Ist – und Sollzustand

Ausgangspunkt für die Analyse ist der Auftrag, „dem Benutzer die Arbeit mit Papier, Bleistift und Radiergummi beim Zeichnen der Baumstrukturen durch ein Programm zu erleichtern".

Istzustand[47]

Wie ist der Benutzer bisher vorgegangen?
Der Benutzer arbeitet mit Papier, Bleistift und Radiergummi. Im 1. Quadranten eines Koordinatensystems zeichnet er einen Punkt ein, z. B. den Punkt (4|2). Zuvor hat er im Beispiel bereits den Punkt (2|3) in den Prioritätssuchbaum eingefügt, wie in der Abbildung 7 ersichtlich ist.

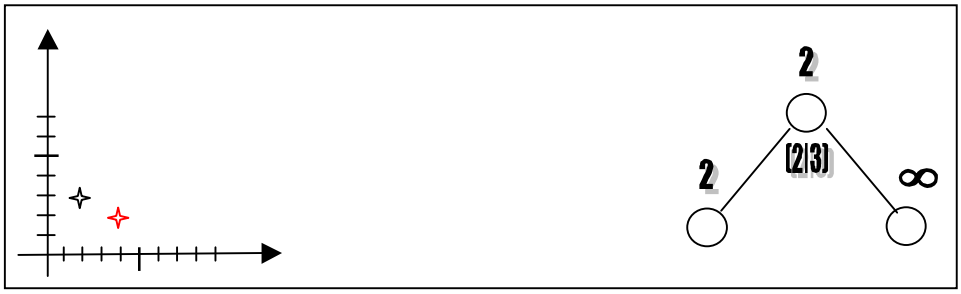


Abbildung 7: Beispiel für die Baumdarstellung (Teil 1)

[46] Die verwendeten UML-Symbole werden im Kapitel A3 des Anhangs erläutert.
[47] Dies stellen wir stellvertretend am Beispiel des Prioritätssuchbaumes vor.

Der Benutzer ergänzt im Prioritätssuchbaum die Knoten für den neuen x-Wert 4 nach den Regeln für den normalen Blattsuchbaum. Das Ergebnis zeigt Abbildung 8.

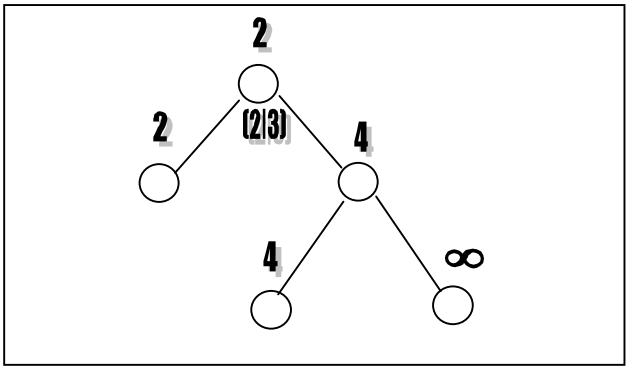


Abbildung 8: Beispiel für die Baumdarstellung (Teil 2)

Jetzt muss der Benutzer den neuen Punkt in den y-Heap einfügen. Dabei kann es zum Verdrängen von zuvor eingefügten Punkten kommen.
Auf dem Weg zum Blatt mit seinem x-Wert 4 wird der neue Punkt in den y-Heap eingefügt. Der Punkt im Wurzelknoten (2|3) weist einen größeren y-Wert auf als der neu einzufügende Punkt (4|2). Deshalb wird der neue Punkt (4|2) in der Wurzel abgelegt und der verdrängte Punkt (2|3) auf seinem Pfad zum Blatt weiter unten eingefügt, wie Abbildung 9 zeigt.

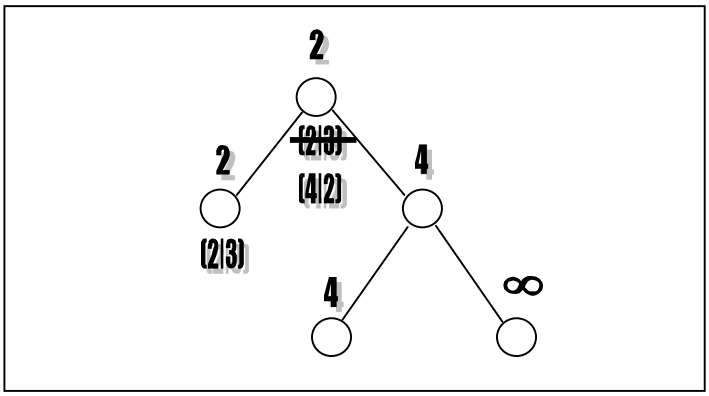


Abbildung 9: Beispiel für die Baumdarstellung (Teil 3)

Der Benutzer radiert verdrängte Punkte aus. Das Streichen der Punkte (Abbildung 9) würde den Prioritätssuchbaum schnell unübersichtlich machen, da die Punkte in einem Knoten häufiger verdrängt werden können.

Eine Schwierigkeit ergibt sich beim Zeichnen der Baumerweiterungen. Will der Benutzer im nächsten Schritt den Punkt (1|5) einfügen, muss er den Baum zunächst um die entsprechenden Knoten erweitern.

Auf schöne Weise geht dies nicht, weil dann ein Knoten direkt über einem bereits vorhandenen Knoten liegen würde, siehe Abbildung 10. Also wählt der Benutzer geringere Knotenabstände, der Baum wird nicht mehr gleichmäßig dargestellt.

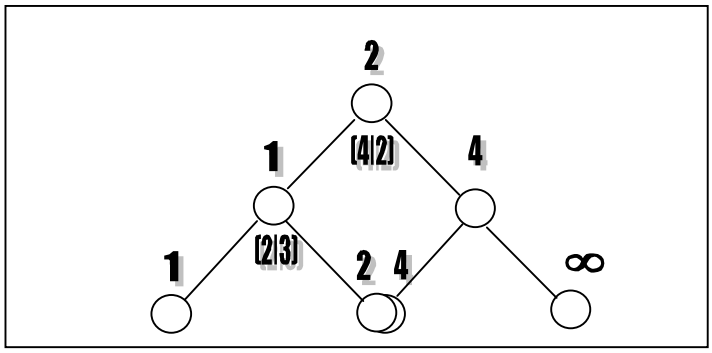


Abbildung 10: Beispiel für die Baumdarstellung (Teil 4)

Wächst der Baum weiter, stellt dies ein immer größeres Problem dar. Der Benutzer kann die neuen Knoten in tieferen Ebenen immer schlechter anordnen. Des öfteren muss er zum Radiergummi greifen und den ganzen Baum ausradieren, um ihn komplett neu mit größeren Abständen zu zeichnen.

Sollzustand

Ein Programm soll dem Benutzer zukünftig das Einfügen von Punkten in einen Prioritätssuchbaum, das Löschen von Punkten aus dem Baum und das Durchführen von Bereichsanfragen ermöglichen.

Dazu werden dem Benutzer verschiedene Varianten des Prioritätssuchbaumes angeboten[48].

Ein Programm muss dem Benutzer also am Bildschirm einen Eingabebereich reduziert auf den 1. Quadranten zur Verfügung stellen, in dem er mit der Maus Punkte eingeben und löschen und einen Anfragebereich einzeichnen kann.

In einem Ausgabebereich soll das Programm den Baum jedes Mal nach einer Aktion in der aktuellen Form anzeigen.

[48] Siehe Kapitel 7.5.1 Datenstruktur

Das Programm soll dabei den beschriebenen Schwierigkeiten begegnen, d.h. es soll die Bäume stets „schön" zeichnen. Die Vorgänge beim Einfügen und Löschen von Punkten und der Bereichsanfrage sollen für den Benutzer nachvollziehbar dargestellt werden.

Eine einheitliche Benutzeroberfläche mit einem Menüsystem und einer Statuszeile soll dem Benutzer ein leichtes Arbeiten mit dem Programm ermöglichen.
Ein Online-Hilfesystem gibt Auskunft zu den Programmmöglichkeiten.

Das Programm soll zusätzlich die aktuellen Punkte in Eingabereihenfolge festhalten, damit der Benutzer jederzeit die Möglichkeit hat, das Gelernte durch Wiederholungen zu festigen.

Gegenüber dem Istzustand ermöglicht das Programm nicht nur die Eingabe von Punkten, um einen Prioritätssuchbaum aufzubauen und zu zeichnen.
Der Benutzer erhält den sukzessiv wachsenden Baum vom Programm und somit eine Kontrollmöglichkeit, ob er den Algorithmus richtig verstanden hat, sich vorher die korrekten neuen Positionen von Knoten und Punkten überlegt hat, der Baum mit dem y-Heap so wie erwartet erweitert wurde.

Zusätzlich wird ein einführendes Demo bereitgestellt. Dieses kann als Lernprogramm eingesetzt werden, um sich den Ablauf der Algorithmen zu verdeutlichen.

7.2 Architektur des Programms

Dem Benutzer wird eine grafische Benutzeroberfläche zum Arbeiten mit dem Programm angeboten.
Die Datenstrukturen, mit denen er arbeitet, müssen einerseits im Speicher verwaltet und andererseits am Bildschirm dargestellt werden.
Benutzeraktionen sollen den Baumzustand verändern. Diese Änderungen erfordern jedes Mal ein Aktualisieren der Baumdarstellung.

Deshalb entscheiden wir uns für den Einsatz des bekannten MVC-Konzeptes[49], das eine saubere Trennung von Daten und deren Ansicht in der grafischen Benutzeroberfläche unterstützt.
Das Modell sind die Baumvarianten, View ist die grafische Oberfläche zum Entgegennehmen der Benutzeraktionen und zur Baumdarstellung.
Controller überwachen die Aktivitäten des Benutzers und den Zustand des Modells. Sie bilden eine enge Einheit mit der grafischen Oberfläche und den dort stattfindenden Ereignissen und schaffen die Verbindung zum Modell und den dort auszuführenden Methoden.

7.3. Gestaltung der Benutzeroberfläche

Beiden Programmteilen, dem 2-d-Baum und dem Prioritätssuchbaum, soll eine einheitliche benutzerfreundliche Bedieneroberfläche zugrunde liegen, die für die unterschiedlichsten Anwendungen aus dem Bereich der algorithmischen Geometrie Ansatz finden kann und die neuen Möglichkeiten von JDK2 zur Gestaltung von Benutzeroberflächen ausnutzt.
Der Programmstart liefert ein Startfenster, das die Auswahl einer Komponente, 2-d-Baum oder Prioritätssuchbaum, ermöglicht.

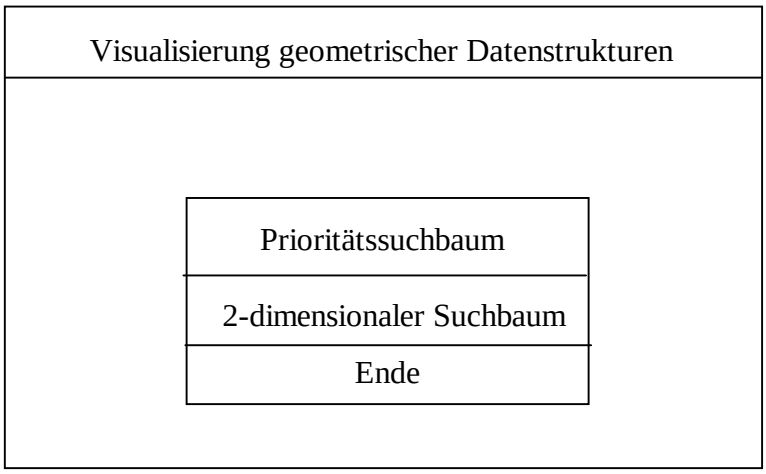


Abbildung 11: Startbildschirm

[49] Beim MVC-Konzept, dem **M**odel-**V**iew-**C**ontroller-Konzept, handelt es sich um ein typisches Entwurfsmuster, siehe auch [GAMM98, Seite 4ff].

Pro Komponente wird dann ein weiteres Fenster als Arbeitsumgebung geöffnet. Abbildung 12 zeigt, dass die Bildschirmgestaltung der Fenster für den 2-d-Baum und den Prioritätssuchbaum einheitlich ist. Nur ihre Menüsysteme differieren in den Unterpunkten der Pulldown-Menüs.

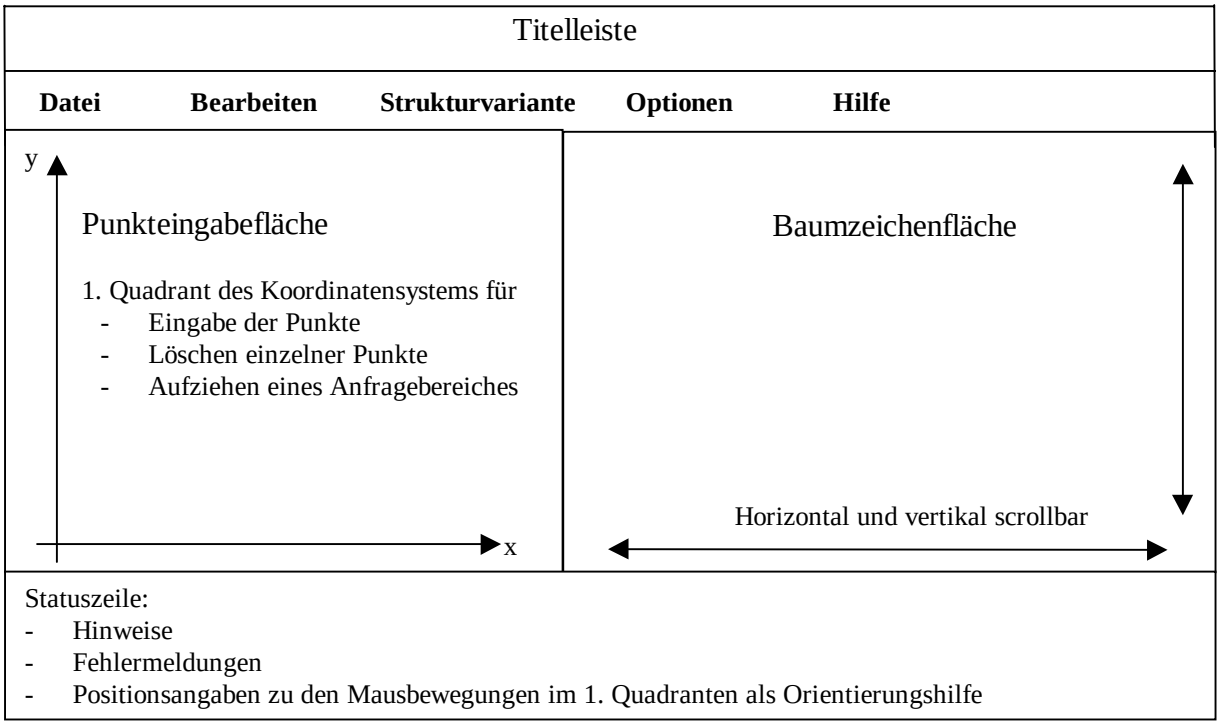


Abbildung 12: Aufteilung des Arbeitsfensters in Bereiche

Der Eingabebereich soll den 1. Quadranten des Koordinatensystems mit einem Punktraster enthalten, in dem der Benutzer Rasterpunkte mit der Maus anklicken kann. Der Eingabebereich muss deshalb Mausaktionen des Benutzers unterstützen.

Der scrollbare Ausgabebereich soll stets den aktuellen Baumzustand anzeigen. In speziellen Fällen muss es möglich sein, Knoten des Baumes anzuklicken, um weitere Informationen zu erhalten. Auch hier müssen dann Mausaktionen unterstützt werden.

Im weiteren erfolgt die Aufteilung der Arbeit in die parallele Entwicklung der spezifischen Teile zum 2-d-Baum, siehe Kapitel 7.4 und zum Prioritätssuchbaum, siehe Kapitel 7.5.

7.4 Elaboration des 2-d-Baums

7.4.1 Datenstruktur

Bei dem k-d-Baum handelt es sich um einen k-dimensionalen Blattsuchbaum, der eine „natürliche Verallgemeinerung“ [FU1840, Seite 118] des eindimensionalen Suchbaums darstellt. Da sich unser Programm **`VisualGeom`** auf Punkte in der Ebene beschränkt, beziehen sich die weiteren Ausführungen nur noch auf den k-d-Baum mit k = 2.

Beim 2-d-Baum werden die Punkte aus der Ebene nach folgenden Kriterien abgelegt:

- Alle Punkte werden in den Blättern eingetragen.
- In den inneren Knoten befinden sich die Splitinformationen.
- Die Splitrichtung wechselt mit jeder Ebene zwischen x- und y-Richtung. Begonnen wird mit der x-Richtung.
- Die Splitinformation wird abhängig von der Splitrichtung aus den x- bzw. y-Werten ausgewählter unmittelbar benachbarter Punkte berechnet.
- Existieren Punkte mit einem Koordinatenwert, der identisch mit dem Splitwert ist, so werden diese Punkte in einem ausgeglichenen 1-d-Baum, also einem eindimensionalen Suchbaum, gespeichert. „Unterhalb eines jeden inneren Knotens ist der eindimensionale Suchbaum für die Punkte auf der Splitgeraden hinzugekommen.“[50] Es entsteht ein ternärer Baum.

Der 2-d-Baum lässt sich in unterschiedlichen Varianten realisieren. Dies hängt davon ab, wie viele Kenntnisse über die Punktmenge vorliegen:

- Die zu betrachtende Punktmenge ist bereits vorgegeben. Sie besteht aus Punkten, die sich in allen Koordinaten voneinander unterscheiden. In diesem Fall ist der Aufbau eines ausgeglichenen 2-d-Baums möglich. Es handelt sich um eine statische Variante des 2-d-Baums.

[50] Siehe [FU1840, Seite 122]

- Die zu betrachtende Punktmenge ist bereits vorgegeben. Sie besteht aber nicht aus Punkten, die sich in allen Koordinaten voneinander unterscheiden. In diesem Fall ist der Aufbau eines erweiterten 2-d-Baums erforderlich. Dieser ist ternär. Es handelt sich wieder um eine statische Variante des 2-d-Baums.
- Die zu betrachtende Punktmenge ist noch nicht vorgegeben. Der 2-d-Baum soll dynamisch aufgebaut werden. Pro auftretendem Punkt ist der 2-d-Baum um einen Knoten zu erweitern. Dazu muss zwischen dem neuen Punkt und dem benachbarten Punkt der Splitwert ermittelt und das Blatt mit dem neuen Punkt ergänzt werden. Der Splitwert kann als arithmetisches Mittel der zugeordneten Punktkoordinaten berechnet oder als Wert aus dem Intervall der zugeordneten Punktkoordinaten festgelegt werden.

7.4.1.1 Der ausgeglichene 2-d-Baum

Die Menge der zu speichernden Punkte wird zunächst in einer Punktliste nach den x-Koordinaten aufsteigend sortiert. Diese Punktliste wird in der Mitte in zwei Punktlisten aufgeteilt. Sie unterscheiden sich in der Anzahl der Elemente um maximal 1. Aus den x-Koordinaten der beiden Punkte in der „Mitte“ wird der Splitwert als arithmetisches Mittel berechnet.
Des Vorgang wird rekursiv abwechselnd mit x oder y als Splitkoordinate durchgeführt, bis nur noch jeweils ein Punkt in der Menge verbleibt.

7.4.1.2 Der allgemeine 2-d-Baum

Die Menge der zu speichernden Punkte wird in einer Punktliste nach den x-Koordinaten aufsteigend sortiert. Existieren Punkte mit identischer x-Koordinate, so werden diese zusätzlich aufsteigend nach ihrer y-Koordinate sortiert. Diese Punktliste wird in der Mitte in zwei Punktlisten aufgeteilt. Sie unterscheiden sich in der Anzahl der Elemente um maximal 1. Aus den x-Koordinaten der beiden Punkte in der „Mitte“ wird der Splitwert als arithmetisches Mittel berechnet.

Existieren Punkte, die den Splitwert als x-Koordinatenwert besitzen, so werden diese in einem eigenen, dem betreffenden inneren Knoten zugeordneten ausgeglichenen 1-d-Baum eingefügt. In diesem Baum sind die Punkte sortiert nach ihrer y-Koordinate eingefügt. Aus den beiden Punktlisten werden alle Punkte, die den Splitwert als x-Koordinatenwert besitzen, entfernt.
Der Vorgang erfolgt rekursiv abwechselnd für x und y als Splitkoordinate.

7.4.1.3 Der dynamische 2-d-Baum

Erfolgt beim Einfügen eines neuen Punktes in den 2-d-Baum nicht sein kompletter Neuaufbau nach dem unter 7.4.1.1 beschriebenen Verfahren, so handelt es sich um eine dynamische Variante des 2-d-Baums. Für die dynamische Variante müssen Algorithmen für das Einfügen und Löschen von Punkten neu entwickelt werden, da diese Variante in der Praxis nicht eingesetzt wird. Sie wird hier aus didaktischen Gründen implementiert, um dem Benutzer das Experimentieren mit dem 2-d-Baum zu ermöglichen. Dabei hat der Benutzer die Auswahl zwischen zwei Fällen. Im ersten Fall wird der jeweilige Splitwert vom Programm als arithmetisches Mittel der zugeordneten Koordinaten der betroffenen Punkte berechnet. Im zweiten Fall muss der Benutzer zuerst die Splitrichtung und dann den Splitwert selbst bestimmen.
Beim Aufbau des Baums kann dieser bis hin zur „Liste“ entarten, denn die AVL-Kriterien für das Balancieren binärer Bäume sind wegen der sich abwechselnden Splitkoordinaten nicht nutzbar. Beim Entfernen eines Punktes kann die Änderung an dem dynamischen Baum im „worst case“ trotzdem seinen kompletten Neuaufbau erforderlich machen. [51]

7.4.2 Use Cases zum 2-d-Baum

Nach den Überlegungen zu der darzustellenden Datenstruktur erarbeiten wir jetzt verschiedene Anwendungsfälle. Sie dienen der Anforderungsermittlung und der groben Abstimmung dessen, was das zukünftige Programm bei der Darstellung von 2-d-Bäumen leisten soll.

[51] Siehe hierzu Kapitel 7.4.3.2

Der Benutzer soll zu den vier verschiedenen Varianten des 2-d-Baumes Punkte einfügen und entfernen und eine Bereichsanfrage durch Festlegen eines Anfragerechtecks initiieren können. Nach allen Aktionen erfolgt das Neuzeichnen des aktuellen 2-d-Baumes. Einführende animierte Demos für die verschiedenen Baumvarianten sollen dem Benutzer das Arbeiten mit dem 2-d-Baum erleichtern.

Unsere Überlegungen führen zu dem allgemeinen „Use Case"–Diagramm von Abbildung 13.

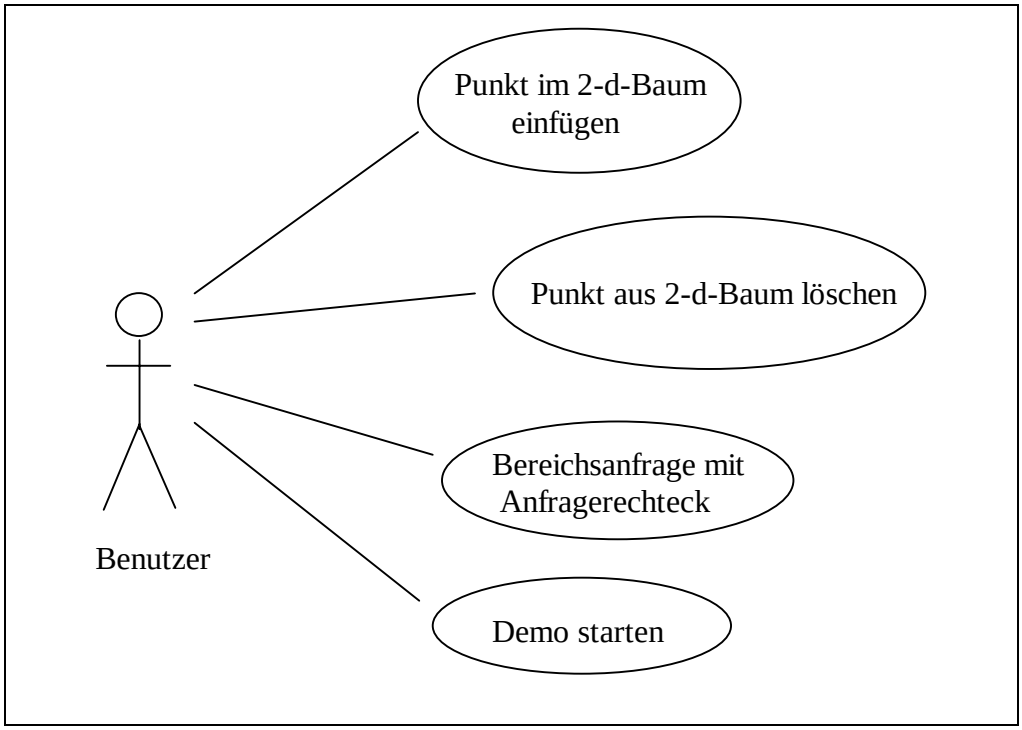


Abbildung 13: Use Case Diagramm[52]

Für die drei verschiedenen Arten von 2-d-Bäumen lassen sich die Fälle „Punkt im 2-d-Baum einfügen" und „Punkt aus 2-d-Baum löschen" in Unterfälle aufteilen. Dies wird im weiteren exemplarisch beschrieben. Auch die Fälle „Bereichsanfrage mit Anfragerechteck" und „Demo starten" werden genauer betrachtet.

[52] Zur Erläuterung der Symbole siehe Tabelle A8

Use Case „Punkt in den dynamischen 2-d-Baum einfügen“

- Der Benutzer klickt mit der linken Maustaste einen Punkt im 1. Quadranten der Ebene an. Dabei werden nur Punkte mit ganzzahligen Koordinatenwerten zugelassen.
- Der Punkt wird in den Eingabebereich eingezeichnet.
- Die Einfügeposition des Punktes im 2-d-Baum wird bestimmt.
- Die Splitkoordinate und der Splitwert werden ermittelt.
- Der Knoten mit dem neuen Splitwert wird in den 2-d-Baum eingefügt.
- Das Blatt mit dem neuen Punkt wird in den Baum eingefügt.
- Die Splitgerade wird in der Ebene dargestellt.
- Im Ausgabebereich wird der 2-d-Baum auf unterschiedliche Weise gezeichnet:
  - Der Baum mit dem neu eingefügten Punkt wird sofort angezeigt.
  - Der Weg beim Einfügen wird dem Benutzer durch eine farblich hervorgehobene Kennzeichnung aller angefassten Punkte veranschaulicht.

Use Case „Punkt aus dem dynamischen 2-d-Baum löschen

- Der Benutzer klickt im 1. Quadranten der Ebene mit der rechten Maustaste einen bereits im 2-d-Baum enthaltenen Punkt an.
- Der Punkt wird aus der Ebene gelöscht.
- Das betroffene Blatt und der zugeordnete Splitknoten werden aus dem Baum entfernt.
- Die Splitgerade wird gelöscht.
- Der 2-d-Baum muss teilweise neu aufgebaut und dann erneut angezeigt werden.

Use Case „Punkte in den ausgeglichenen 2-d-Baum einfügen“

- Der Benutzer klickt mit der linken Maustaste alle für den Baumaufbau auszuwählenden Punkte im 1. Quadranten der Ebene an. Dabei werden nur Punkte mit ganzzahligen Koordinatenwerten zugelassen, die sich in allen Koordinaten voneinander unterscheiden.
- Die Punkte werden in den Eingabebereich eingezeichnet.
- Sind alle Punkte eingegeben worden, veranlasst der Benutzer den Aufbau des ausgeglichenen 2-d- Baums.
- Von Baumebene zu Baumebene wechselt die Splitrichtung, beginnend bei der x-Koordinate. Die jeweiligen Splitwerte werden berechnet und der Splitknoten in den Baum eingefügt.
- Die Punkte werden in den Blättern eingetragen.
- Im Ausgabebereich wird der 2-d-Baum auf unterschiedliche Weise dargestellt:
  - Der 2-d-Baum und die Splitgeraden werden sofort angezeigt.
  - Der 2-d-Baum und die Splitgeraden werden durch Benutzeraufforderung schrittweise Ebene für Ebene gezeichnet.
  - Die Abläufe beim Baumaufbau werden schrittweise in einer Animation veranschaulicht.

Use Case „Punkte in den allgemeinen 2-d-Baum einfügen“

- Der Benutzer klickt mit der linken Maustaste alle Punkte im 1. Quadranten der Ebene an. Dabei werden nur Punkte mit ganzzahligen Koordinatenwerten zugelassen. Das mehrfache Eingeben eines Punktes ist nicht möglich.
- Die Punkte werden in den Eingabebereich eingezeichnet.
- Sind alle Punkte eingegeben worden, veranlasst der Benutzer den Aufbau des allgemeinen 2-d-Baums.
- Von Baumebene zu Baumebene wechselt beim Aufbau die Splitrichtung. Es werden die jeweiligen Splitwerte berechnet. Dabei wird berücksichtigt, ob Punkte vorhanden sind, deren Koordinatenwerte mit dem Splitwert übereinstimmen.
- Für Punkte, die auf der Splitgeraden liegen wird ein separater 1-d-Baum aufgebaut, der dem entsprechenden Splitknoten zugeordnet ist.
- Die Punkte werden in den Blättern eingetragen.
- Im Ausgabebereich wird der 2-d-Baum auf unterschiedliche Weise gezeichnet:
  - Der 2-d-Baum und die Splitgeraden werden sofort angezeigt.
  - Der 2-d-Baum und die Splitgeraden werden nach Benutzeraufforderung schrittweise Ebene für Ebene gezeichnet.
  - Die Abläufe beim Baumaufbau werden schrittweise in einer Animation veranschaulicht.

Use Case „Rechteckanfrage durchführen“

- Im Eingabebereich zieht der Benutzer mit der linken Maustaste das Rechteck für die Anfrage auf.
- Der 2-d-Baum wird abhängig von dem ausgewählten x- und dem ausgewählten y-Intervall durchsucht. Dabei müssen alle Knoten angefasst werden, für die der Durchschnitt aus zugeordnetem Splitbereich und Anfragerechteck nicht leer ist.
- Die bei der Suche aufgefundenen Punkte und alle besuchten Knoten werden farblich gekennzeichnet in den 2-d-Baum eingezeichnet.

Use Case „Demo aufrufen“

- Der Benutzer wählt im Hilfemenü eine der beiden angebotenen Demos zum dynamischen oder ausgeglichenen 2-d-Baum.
- Das Programm zeigt schrittweise den Baumaufbau beim Einfügen von Punkten. Im dynamischen Fall wird ein Punkt gelöscht.
- Es wird ein Anfragerechteck im Eingabebereich eingezeichnet. Dann erfolgt die Markierung aller angefassten und aller im Anfragerechteck enthaltenen Punkte.
- Über eine Schaltfläche kann das Demo jederzeit vom Benutzer definiert abgebrochen werden.

7.4.3 Verfeinerte Analyse mit Aktivitätsdiagrammen

Einen ersten Übergang von der Analyse, also dem

„Was soll gemacht werden?”

zum Entwurf, also dem

„Wie soll dies geschehen?”

bieten Aktivitätsdiagramme[53]. Sie stellen verfeinerte Abläufe der in den Use-Cases aufgeführten Aktionen dar und bringen Möglichkeiten ins Spiel, wie diese Forderungen erfüllt werden können. Dies soll hier für die Anwendungsfälle von Kapitel 7.4.2 betrachtet werden. Dabei erfolgen auch Überlegungen, wie die Visualisierung der Baumalgorithmen durchgeführt werden sollte.

7.4.3.1 Punkt in den dynamischen 2-d-Baum einfügen

Wenn der Benutzer einen Punkt im 1. Quadranten der Ebene mit der Maus angeklickt hat, muss der Punkt von den Bildschirmkoordinaten in die Weltkoordinaten umgerechnet werden. Dabei ist das jeweils eingestellte Raster zu berücksichtigen. Den Zusammenhang zeigt Abbildung 14.

[53] Sinngemäß nach [OEST98]

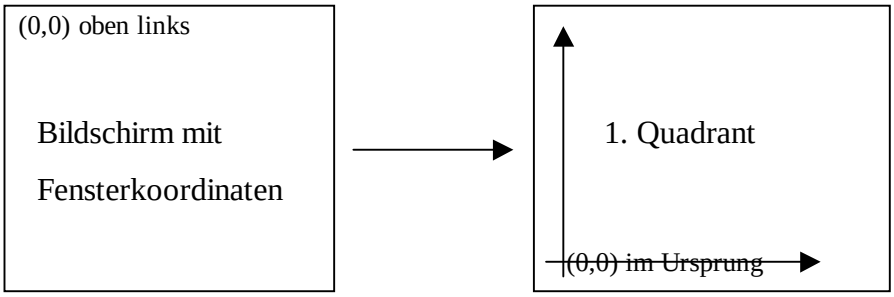


Abbildung 14: Koordinatentransformation

Bevor der ermittelte Punkt in den dynamischen 2-d-Baum eingefügt werden kann, muss geklärt werden, ob es sich bei diesem um einen für die ausgewählte Strukturvariante zulässigen Punkt handelt. Folgende Rahmenbedingungen stellen die Voraussetzungen für die Übernahme in den Baum dar:

- Der Punkt ist noch nicht im 2-d-Baum eingetragen.
- Der Punkt unterscheidet sich von den bereits eingetragenen Punkten in beiden Koordinaten.

Da auch der erneute Aufbau des 2-d-Baumes mit **`VisualGeom`** möglich sein soll, wird als Hilfsdatenstruktur eine Punktliste genutzt. Diese soll alle bereits im 2-d-Baum enthaltenen Punkte in der Eingabereihenfolge festhalten. Damit ergibt sich eine einfache Möglichkeit, diese Liste zur Kontrolle der Voraussetzungen für das Eintragen des neuen Punktes zu durchsuchen. Handelt es sich bei dem angeklickten Punkt um einen, der nicht den Voraussetzungen genügt, soll eine Fehlermeldung den Benutzer auf die Falscheingabe hinweisen.
Andernfalls erfolgt das Anzeigen des Punktes, das Anhängen an das Ende der Punktliste, das Ermitteln des Splitwertes[54], sein Einfügen in den dynamischen 2-d-Baum, die Ermittlung von Anfangs- und Endpunkt der neuen Splitgeraden und die Darstellung der Splitgeraden.
Dem Benutzer muss dabei deutlich werden, welche Änderungen am 2-d-Baum vorgenommen werden.[55]

Das Ergebnis unserer Überlegungen zeigt das Aktivitätsdiagramm von Abbildung 15.

[54] Der Splitwert kann als arithmetisches Mittel berechnet oder durch Benutzereingabe bestimmt werden.
[55] Siehe hierzu auch „Überlegungen zur Visualisierung beim Einfügen eines Punktes“ auf der Seite 47

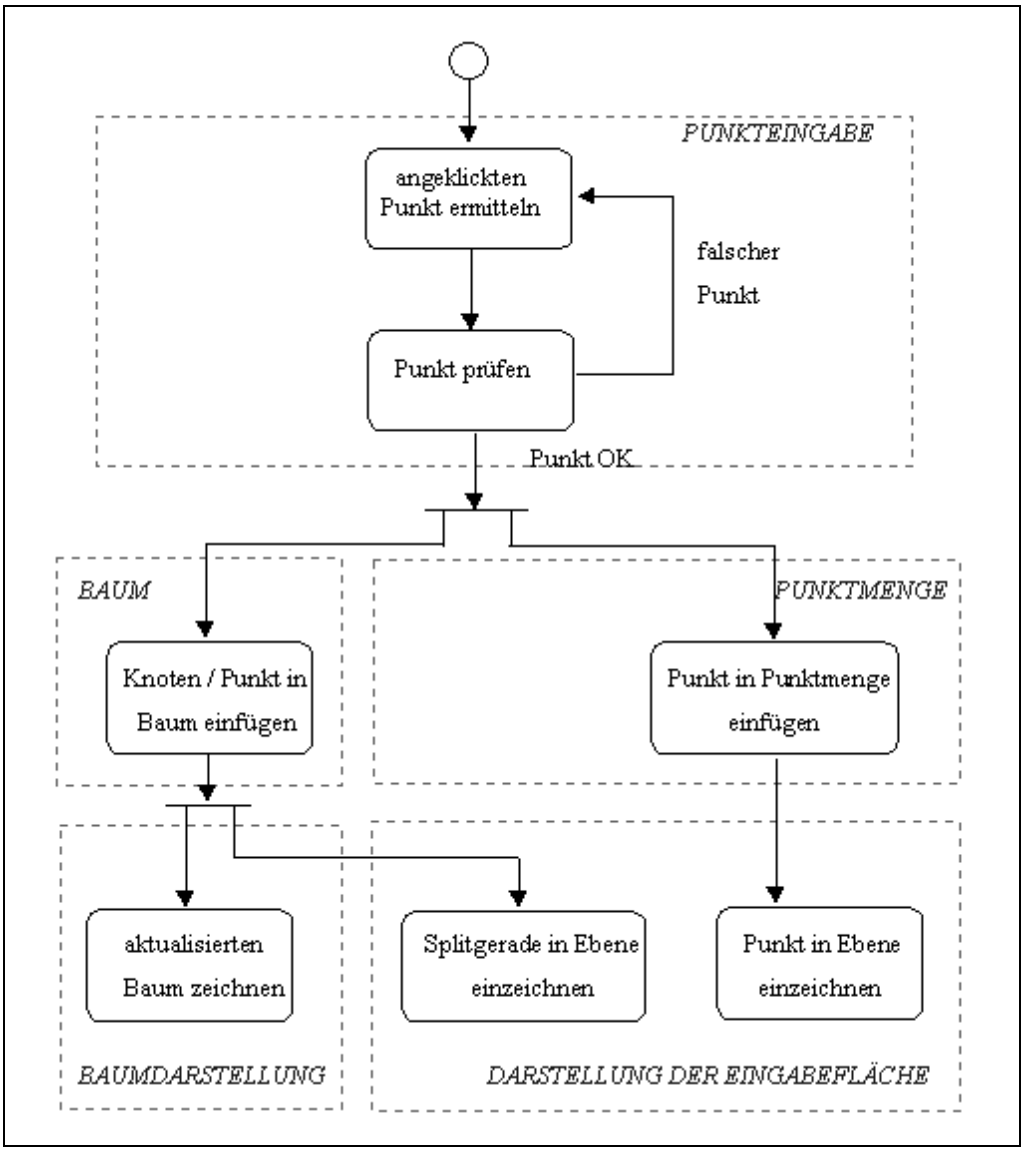


Abbildung 15: Aktivitätsdiagramm „Punkt in dynamischen 2-d-Baum einfügen“[56]

Allgemeine Überlegungen zur Baumdarstellung

Der 2-d-Baum wird nach dem Einfügen eines weiteren Punktes in seiner neuen Form angezeigt. Dabei sind im Ausgabebereich die Kanten, die Knoten, die Splitwerte und die Punkte in schwarz darzustellen. Im Eingabebereich werden die Punkte als kleine schwarze Rechtecke und die Splitgeraden in rot eingezeichnet.

[56] Zur Erläuterung der Symbole siehe Tabelle A9

Visualisierung beim Einfügen eines Punktes

- **Sofortiger Aufbau**

  Der dynamische 2-d-Baum wird unmittelbar nach der Eingabe des neuen Punktes aktualisiert. Auch wird sofort die neue Splitgerade im Eingabebereich dargestellt.

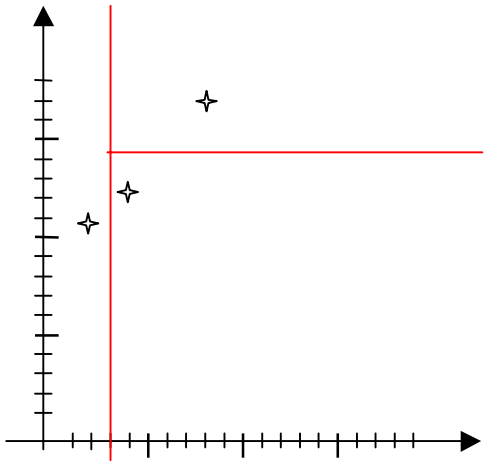
- **Animiert**

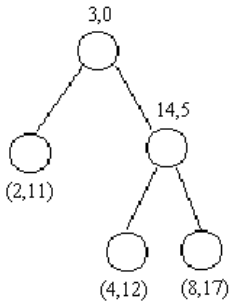
  Im Gegensatz zum Einfügen und Löschen im Prioritätssuchbaum erscheint uns beim Einfügen eines Punktes in den dynamischen 2-d-Baum eine Dynamisierung zur Veranschaulichung nicht erforderlich.

  Der Benutzer weiß aufgrund der Cursorposition im Eingabebereich, welcher Punkt neu ist und dass der Splitbereich, in dem sich bereits ein weiterer Punkt befindet, nun zwischen diesem und dem neuen Punkt aufgeteilt wird. Aus der Baumdarstellung ist außerdem entnehmbar, wo sich das zugeordnete Blatt dieses Punktes im Baum befindet und damit, wo die Veränderung im Baum durchgeführt wird.

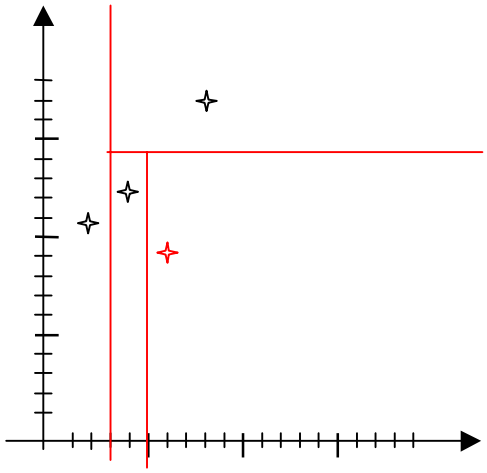
  Wir haben uns deshalb entschlossen, in diesem Fall nur kenntlich zu machen, welche Knoten beim Einfügen angefasst werden müssen. Diese Knoten werden durch zusätzliche rote Kreise gekennzeichnet. Die beiden Punkte, die von der Veränderung im 2-d-Baum betroffen sind, werden außerdem rot dargestellt.

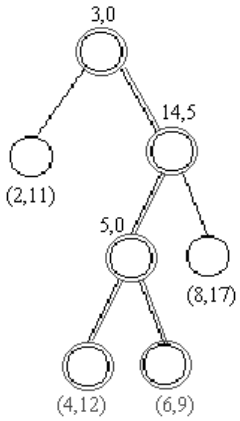
  Mit dieser Visualisierung erhält der Benutzer die Möglichkeit, die Baumänderungen aufgrund seiner Punkteingabe zu analysieren. Auch erkennt er dabei, welche Knoten durch den implementierten Algorithmus angefasst werden. Damit lässt sich sogar eine Vermutung über die Laufzeit beim Einfügen eines Punktes treffen.





1. Schritt: Ausgangssituation mit den bereits eingefügten Punkten (2|11), (4|12) und (8|17)





2. Schritt: Einfügen des Punktes (6|9)

Abbildung 16: Beispielskizzen zum Einfügen des Punktes (6|9)

Wie aus Abbildung 16 ersichtlich, liegt der neu einzufügende Punkt (6|9) im Splitbereich des Punktes (4|12). Dies bedeutet, dass das Blatt (4|12) am Knoten mit dem Splitwert y=14,5 abzuhängen ist. Dort wird jetzt der Knoten mit dem neuen Splitwert x=5, der sich als arithmetisches Mittel der x-Koordinaten 4 und 6 ergibt, angehängt. An diesem neuen Knoten wird links das Blatt mit dem Punkt (4|12) und rechts das mit dem Punkt (6|9) angeheftet. Die Splitgerade mit dem Splitwert x=5 wird im Eingabebereich ab der übergeordneten Splitgeraden mit y=14,5 eingezeichnet.

Durch die farbliche Kennzeichnung sind somit alle beim Baumaufbau entstandenen Änderungen ohne Probleme erkennbar.

7.4.3.2 Punkt aus dem dynamischen 2-d-Baum löschen

Im Eingabebereich wählt der Benutzer den zu löschenden Punkt mit der rechten Maustaste aus. Befindet sich die Maus über einem bereits vorhandenen Punkt, muss dieser sowohl aus der Punktliste als auch dem 2-d-Baum entfernt werden. Dies bedeutet aber, dass der zugeordnete Splitknoten auch nicht mehr erforderlich ist und damit auch die zugeordnete Splitgerade zu löschen ist.

Wegen des Wechsels der Splitkoordinate zwischen „x“ und „y“ von Baumebene zu Baumebene lässt sich der Löschvorgang nicht auf das Löschen eines Blattes in einem binären Suchbaum zurückführen. Deshalb folgen hier nun grundsätzliche Überlegungen zu einem möglichen Löschalgorithmus. Nur so können wir die Design-Entscheidung treffen, ob das Löschen im Rahmen unserer Software-Entwicklung überhaupt implementierbar ist.
In der Literatur fanden wir bis auf das schwache Entfernen [FU1840, Seite 122] keinen Hinweis über einen Löschalgorithmus für einen 2-d-Baum. Dieses schwache Verfahren verbietet sich hier allerdings. Der Knoten im Baum ließe sich ohne Probleme geeignet kennzeichnen. Im Rahmen der Visualisierung soll aber nicht nur die Datenstruktur, sondern auch der 1. Quadrant mit den Punkten und den Splitgeraden dargestellt werden. Beim schwachen Entfernen würde es dort zu unüblichen Darstellungen kommen. Der betroffene Bereich wäre durch das Entfernen des Punktes und der zugeordneten Splitgeraden i. Allg. kein Rechteck mehr.[57]

Um eine Idee zum zu entwickelnden Löschalgorithmus zu erhalten, werden nachfolgend zunächst einige mögliche Löschfälle betrachtet.

Im Fall von Abbildung 17 existiert noch kein Splitwert und damit auch noch keine Splitgerade. In der Punktmenge befindet sich nur ein Punkt. Der 2-d-Baum besteht aus einem Blatt.
In diesem trivialen Fall müssen nur die Punktliste geleert und das Blatt aus dem Baum entfernt werden.

[57] Aus diesem Grunde haben wir uns auch dazu entschlossen, auf das Löschen im ausgeglichenen und allgemeinen 2-d-Baum zu verzichten. Beim Löschen und auch beim Einfügen weiterer Punkte wird ein kompletter Neuaufbau des 2-d-Baums durchgeführt.

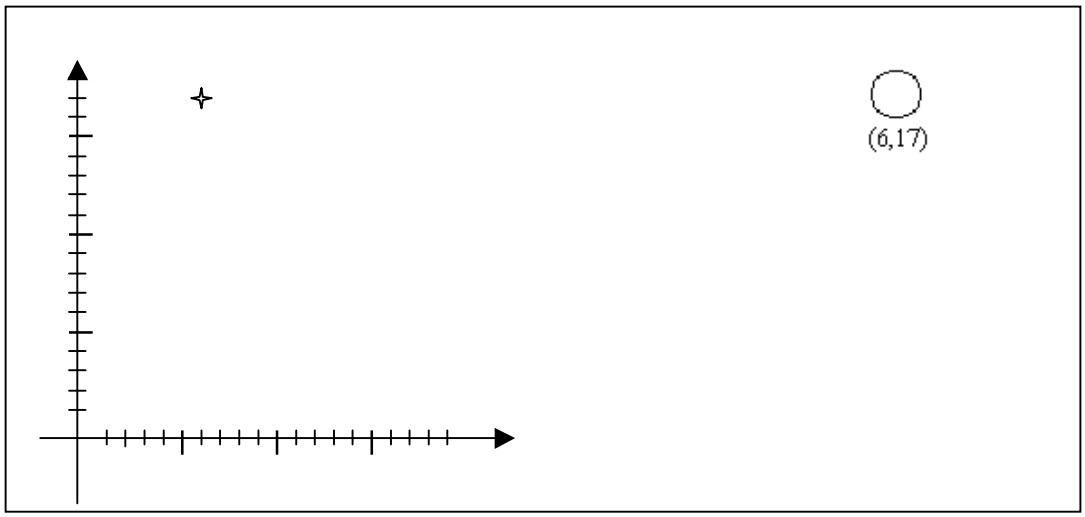


Abbildung 17: 2-d-Baum mit einem Blatt

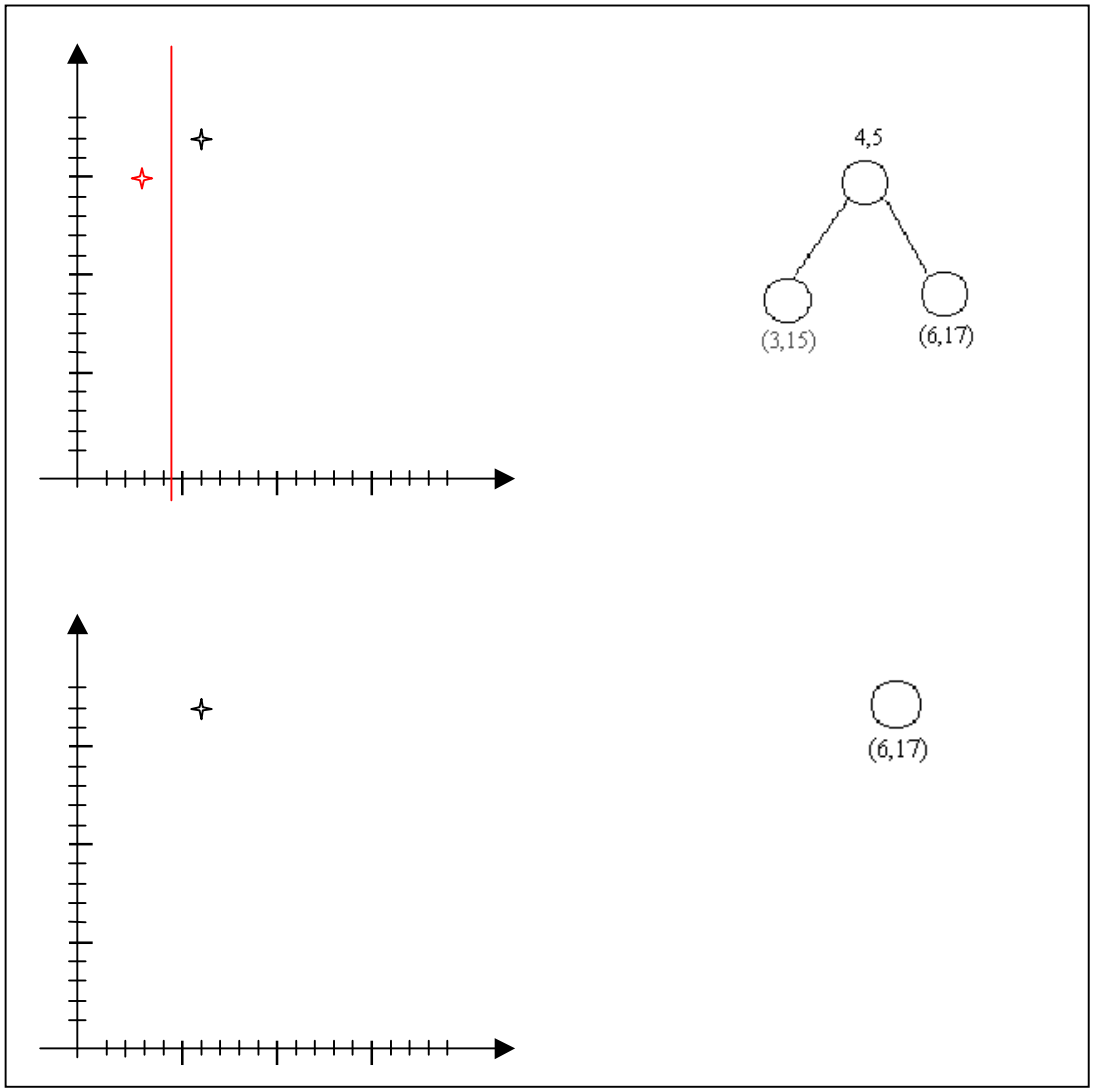


Abbildung 18: 2-d-Baum mit zwei Blättern

Im Fall von Abbildung 18 ergibt sich beim Löschen eines der beiden Punkte, z. B. (3|15), kein Problem. Der ausgewählte Punkt muss aus der Punktliste entfernt werden. Der neu aufzubauende 2-d-Baum besteht nur noch aus einem Blatt. Also sind der Splitknoten mit x=4,5 und das Blatt mit dem Punkt (3|15) aus dem 2-d-Baum und die zugeordnete Splitgerade aus dem Eingabebereich zu entfernen.

Auch dieser Fall scheint problemlos implementierbar zu sein.

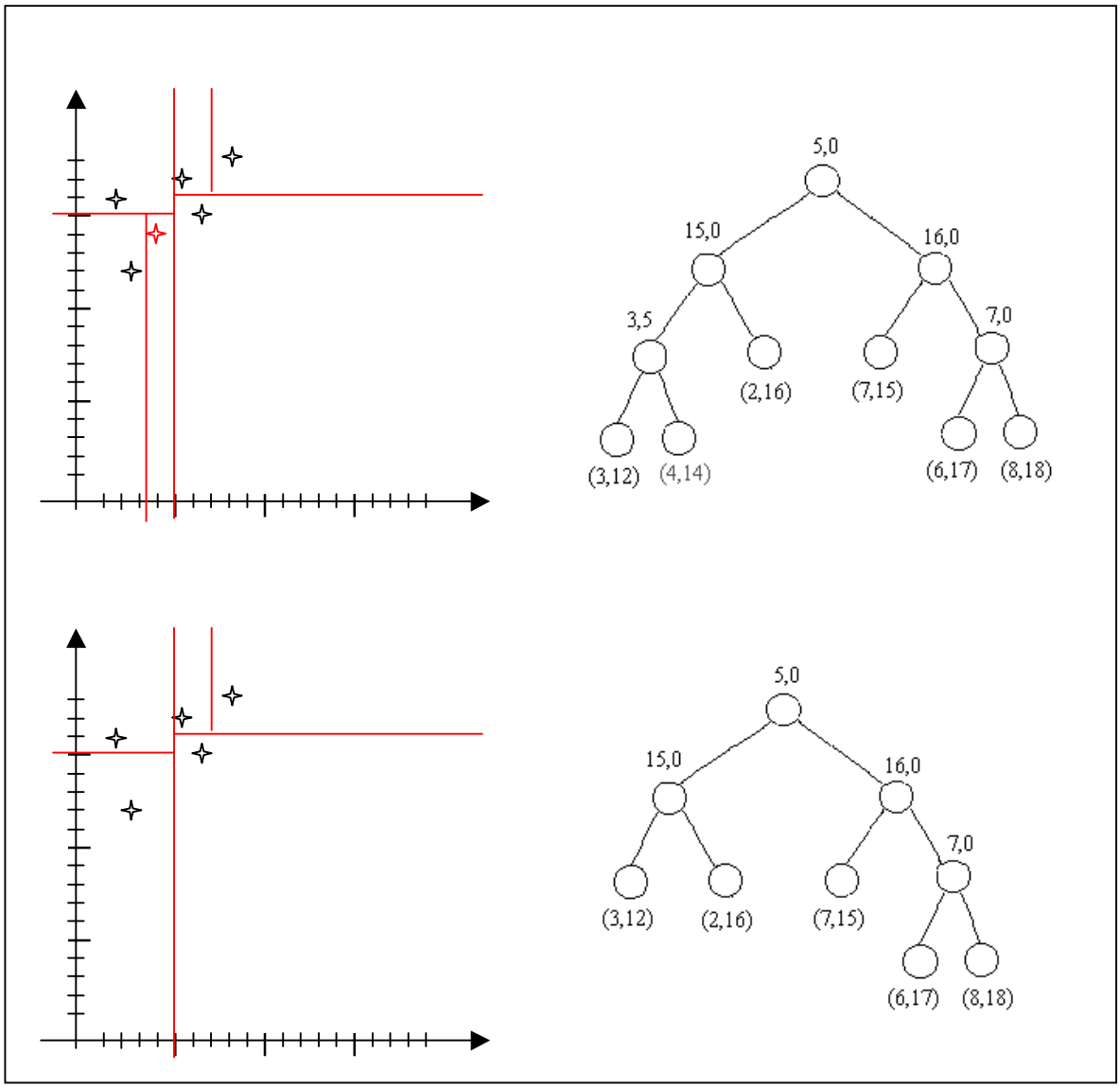


Abbildung 19: 2-d-Baum mit zwei Blättern an einem Splitknoten

In Abbildung 19 wird mit (4|14) ein Blatt gelöscht, an dessen Splitknoten mit x=3,5 sich das weitere Blatt mit (3|12) befindet. Dies bedeutet, dass mit dem zu löschenden Blatt auch der Knoten mit der Splitinformation und damit die Splitgerade zu löschen ist. Das verbleibende Blatt mit (3|12) muss eine Ebene höher rücken, Nachfolger des Knotens mit dem Splitwert y=15 werden. Der Splitwert y=15 stellt nicht das arithmetische Mittel von $y_1 = 12$ und $y_2 = 16$ dar. Eine Korrektur des Splitwertes ist allerdings nicht zwingend erforderlich, da als Splitwert jedes y aus dem offenen y-Intervall, also $y \in]12;16[$, ausgewählt werden darf.

Damit scheint auch dieser Fall problemlos implementierbar.

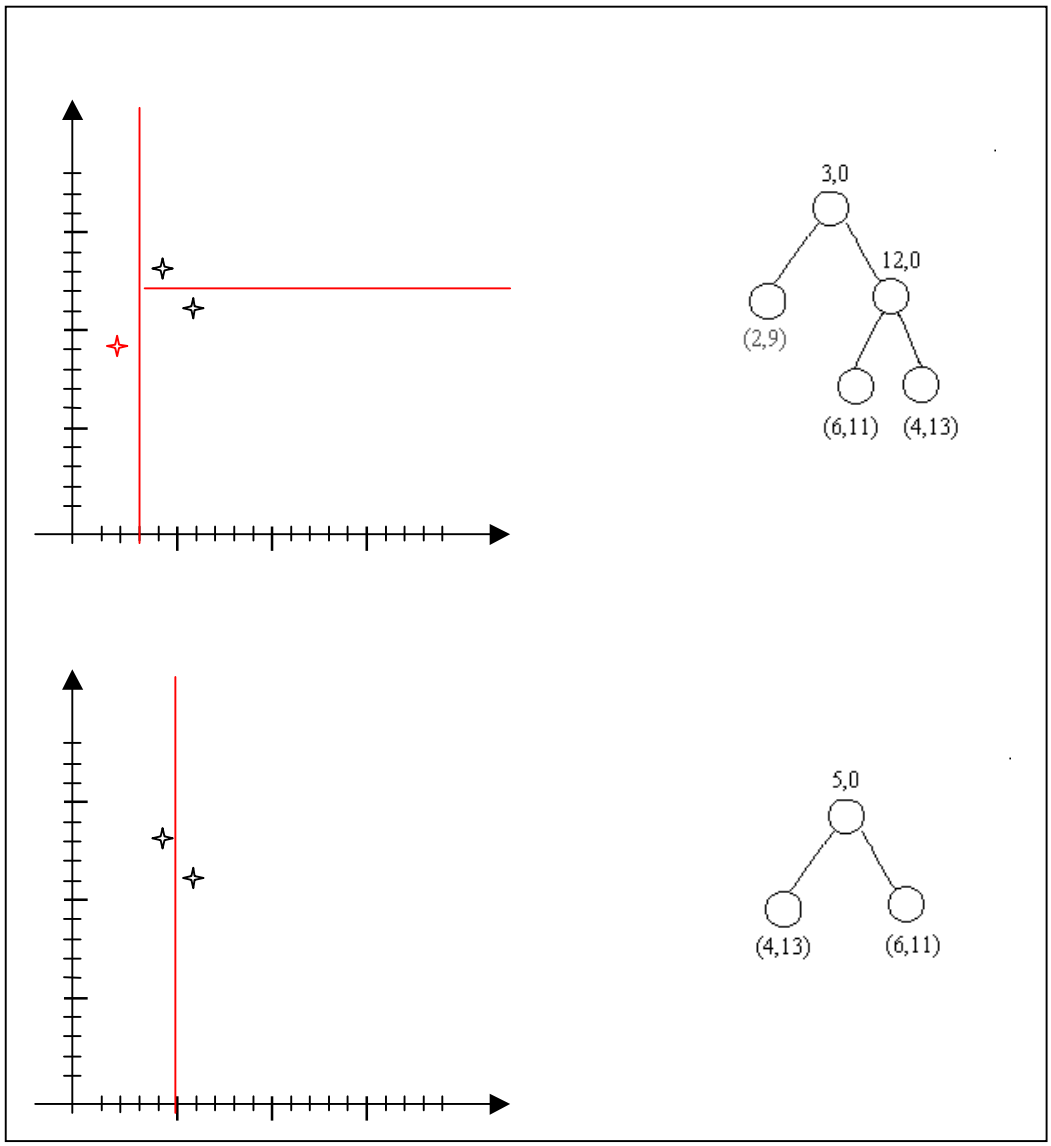


Abbildung 20: 2-d-Baum mit einen Blatt am Wurzelknoten

In Abbildung 20 befindet sich am linken Ast der Wurzel ein Blatt und am rechten Ast ein kompletter Teilbaum. Wird nun der Punkt (2|9) gelöscht, so ist der Wurzelknoten mit seinem Splitwert überflüssig. Da sich im 2-d-Baum die Splitkoordinaten zyklisch abwechseln, aber stets mit den x-Koordinaten bei der Berechnung des ersten Splitwertes begonnen wird, müssen in diesem Fall beide Splitknoten gelöscht und die verbleibenden beiden Punkte in einen neu aufzubauenden Baum eingefügt werden.

Dies bedeutet verallgemeinert, dass stets ein kompletter Neuaufbau mit der verbleibenden Punktliste erforderlich wird, wenn im Baum ein Blatt gelöscht wird, das unmittelbar am Wurzelknoten angeheftet ist.

Dies zu implementieren, stellt kein Problem dar. Die Punkte aus der Punktliste werden in der Eingabereihenfolge nacheinander in einen neuen 2-d-Baum eingefügt.

Im „worst case“ führt dieser Neuaufbau des dynamischen 2-d-Baums zu einer „Liste“. Damit ergibt sich $O(n)$ für die Laufzeit dieses Löschalgorithmus.

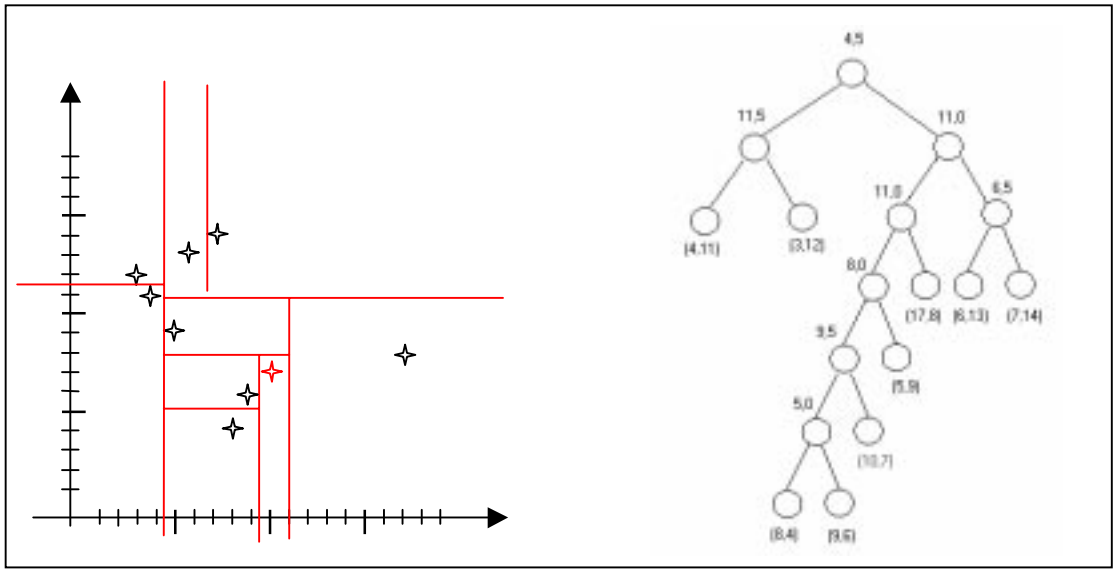


Abbildung 21: Löschen in einem beliebigen 2-d-Baum

Im Fall von Abbildung 21 kann einer von insgesamt acht Punkten gelöscht werden. Dabei entspricht das Löschen der Punkte (8|4), (9|6), (6,13), (7|14), (4|11) und (3|12) dem in Abbildung 19 behandelten Fall.

Wird der Punkt (10|7) gelöscht, so führt dies dazu, dass der linke Teilbaum am Splitknoten y=8 ohne den gelöschten Punkt komplett neu aufgebaut werden muss.
Dies bedeutet, dass die verbleibenden Punkte im linken Teilbaum, also (8|4) und (9|6), zunächst in einer Punktliste als Hilfsdatenstruktur zu sichern sind.

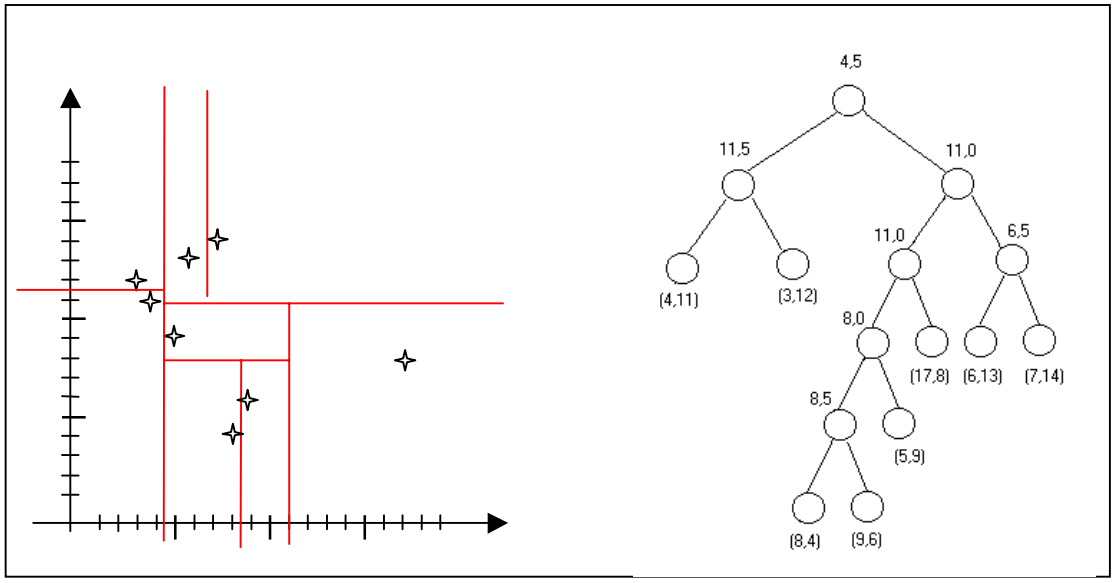


Abbildung 22: 2-d-Baum nach Löschen des Punktes (10|7)

Das Löschen des Punktes (5|9) aus Abbildung 21 bewirkt, dass die Punkte (8|4), (9|6) und (10|7) aus dem Baum entfernt und in einer Punktliste zwischengespeichert werden müssen. Am Knoten mit dem Splitwert x=11 kann dann der neue Teilbaum aufgebaut werden.
Entsprechende Überlegungen gelten auch für das Löschen des Punktes (17|8).

> Abgesehen von den in der Abbildung 18 und der Abbildung 19 dargestellten Sonderfällen sind stets umfangreichere Operationen beim Löschen eines Punktes erforderlich.
> Ab dem gelöschten Punkt muss der betreffende Teilbaum neu aufgebaut werden. Dabei wird der Neuaufbau um so signifikanter, je näher der zu löschende Punkt in der Nähe der Wurzel liegt.

Damit erscheint es uns möglich, im Rahmen der Diplomarbeit den gewünschten Löschalgorithmus für den dynamischen 2-d-Baum zu implementieren.

7.4.3.3 Punkte in den ausgeglichenen 2-d-Baum einfügen

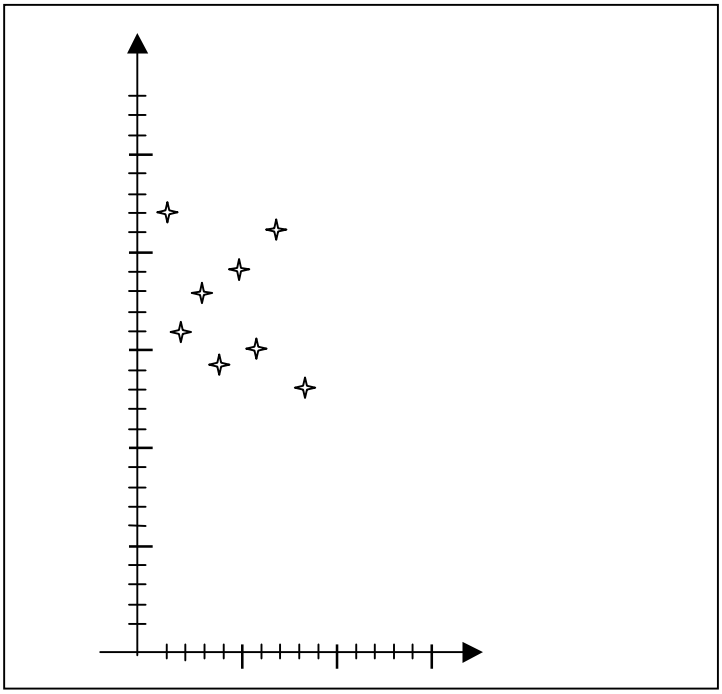


Abbildung 23: Aufbau eines ausgeglichenen 2-d-Baums (Teil 1)

Der ausgeglichene 2-d-Baum wird erst dann aufgebaut, wenn alle Punkte ausgewählt wurden (siehe Abbildung 23). Die Punkte werden nach ihrer x-Koordinate sortiert in die Punktliste eingefügt.

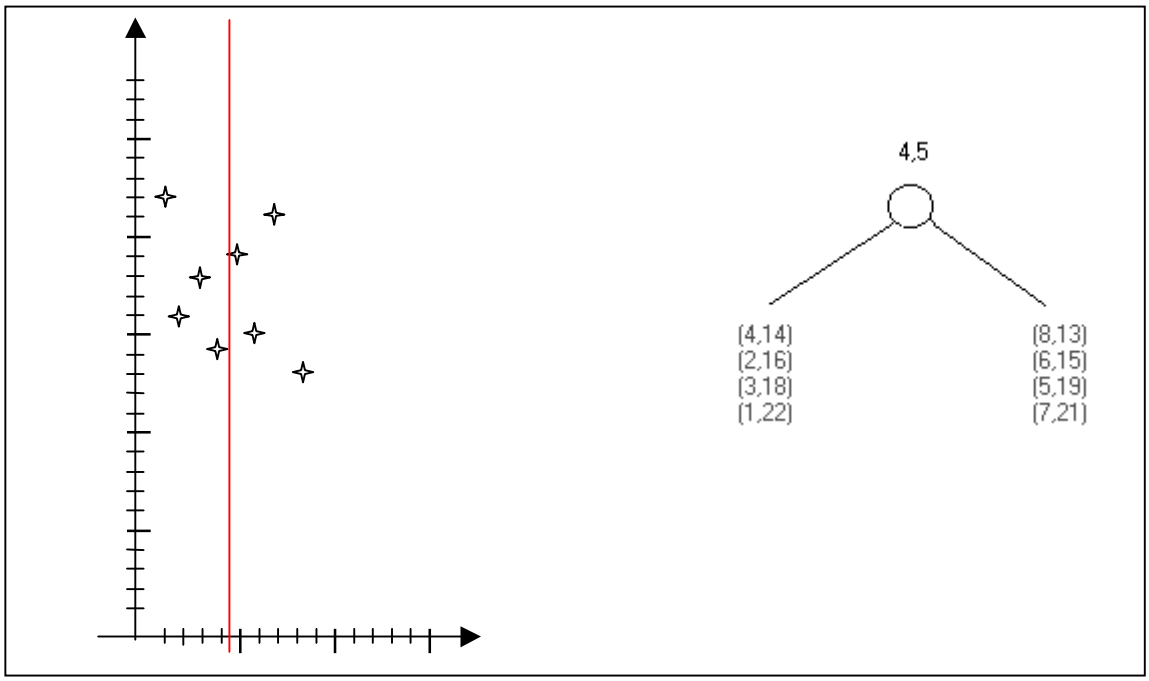


Abbildung 24: Aufbau eines ausgeglichenen 2-d-Baums (Teil 2)

Der erste Splitwert ergibt sich als arithmetisches Mittel der beiden x-Koordinaten der „mittleren“ Punkte in der sortierten Punktliste. Hier ist es x=4,5. Dann wird die Punktliste in zwei gleich große nach der y-Koordinate sortierte Punktlisten aufgeteilt (Abbildung 24).

Mit diesen nach den y-Koordinaten sortierten Punktlisten wird analog verfahren. Nur werden jetzt für die Ermittlung der Splitwerte die entsprechenden y-Koordinaten verwendet. Es ergibt sich der Teilbaum von Abbildung 25. An den Splitknoten sind nun die aufgeteilten, wiederum nach der x-Koordinate sortierten Punktlisten angeheftet.

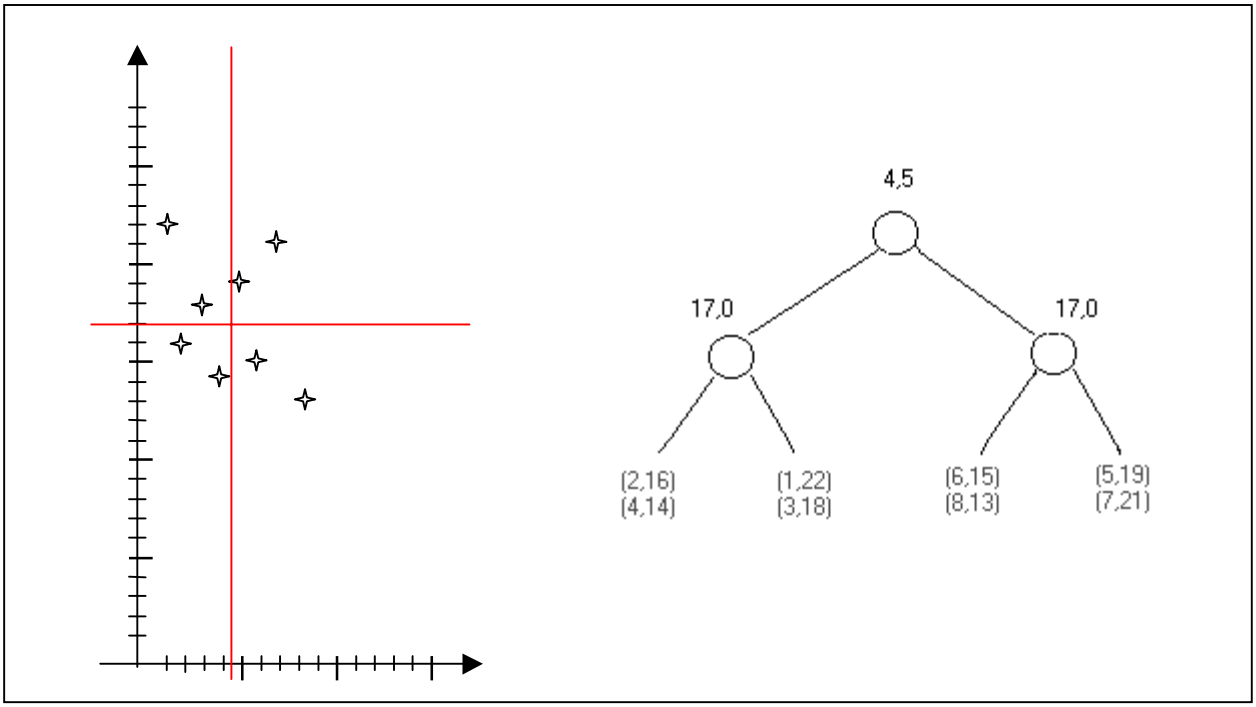


Abbildung 25: Aufbau eines ausgeglichenen 2-d-Baums (Teil 3)

Entsprechend wird im nächsten Schritt verfahren. Nun werden für die Ermittlung der Splitwerte wieder die x-Koordinaten verwendet. Es ergibt sich die Abbildung 26. An jedem Splitknoten befindet sich nun eine Punktliste mit einem Punkt. Im letzten Schritt kann also der Baumaufbau vervollständigt werden, indem an den zugeordneten Splitknoten nur noch die Blätter mit den Punkten angeheftet werden, siehe Abbildung 27.

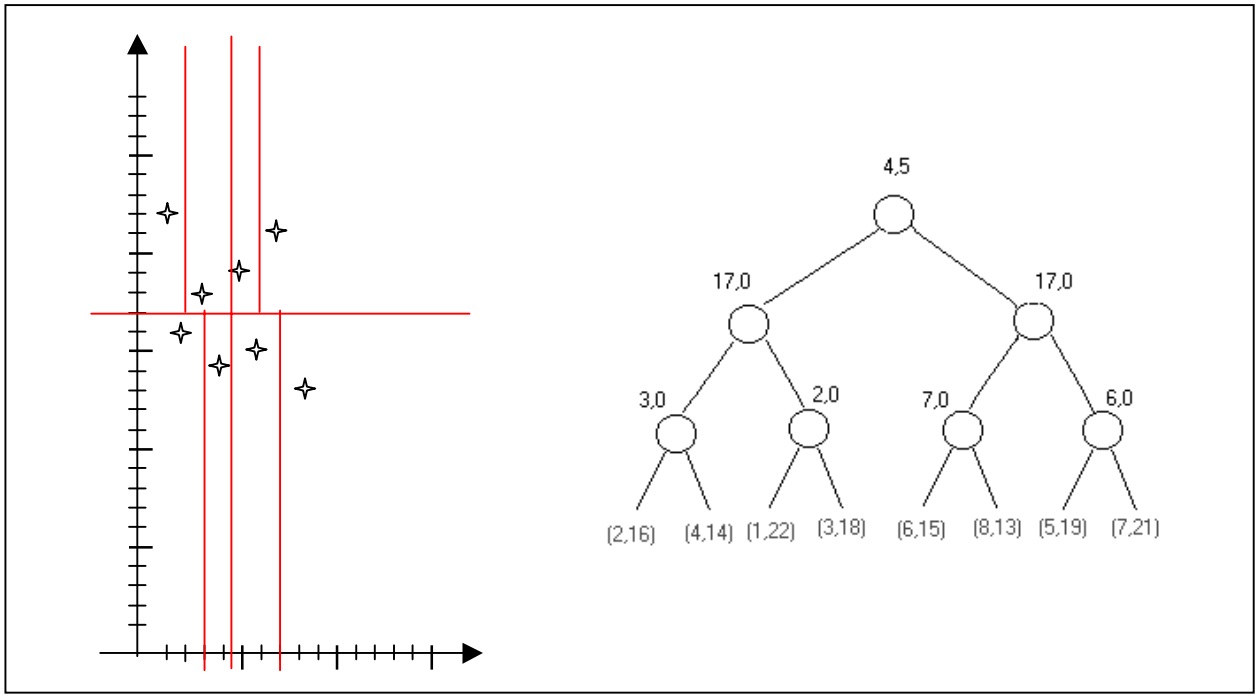


Abbildung 26: Aufbau eines ausgeglichenen 2-d-Baums (Teil 4)

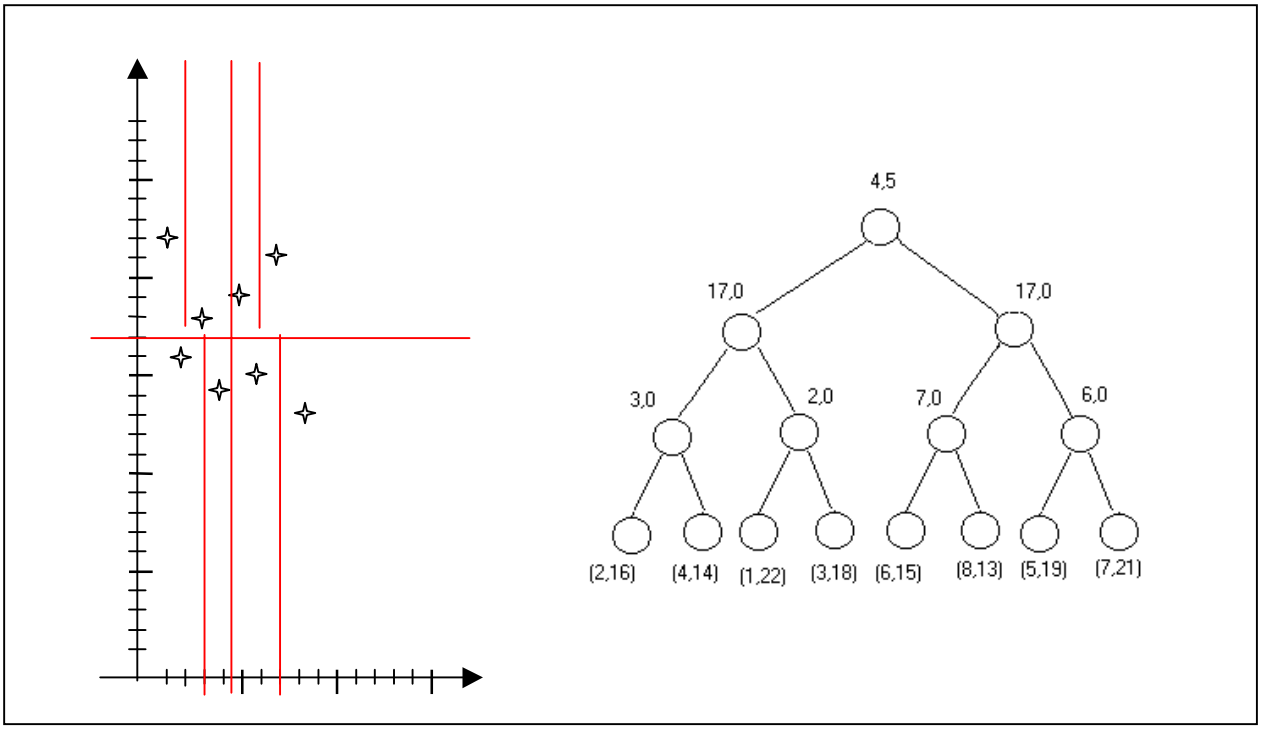


Abbildung 27: Aufbau eines ausgeglichenen 2-d-Baums (Teil 5)

Visualisieren des Aufbaus eines ausgeglichenen 2-d-Baums

- **Sofortiger Aufbau**
  Nach der Eingabe der Punkte und dem Betätigen der Schaltfläche „Baum zeichnen“ werden der 2-d-Baum und alle Splitgeraden unmittelbar eingezeichnet.

- **Schrittweise**
  Nach der Eingabe der Punkte und dem Betätigen der Schaltfläche „Baum zeichnen“ wird der Wurzelknoten mit den beiden nach y-Koordinaten sortierten Punktlisten dargestellt. Der Benutzer kann den weiteren Baumaufbau sukzessive Ebene für Ebene vornehmen lassen (siehe Abbildung 23 bis Abbildung 26). Dafür wird eine Schaltfläche „weiter“ zur Verfügung gestellt.

- **Animiert**
  Während unter „schrittweise“ der nächste Schritt durch den Benutzer erfolgt, geschieht dies hier automatisiert. Der Vorgang kann dabei zu jeder Zeit definiert abgebrochen werden.

7.4.3.4 Rechteckanfrage durchführen

Der Benutzer zieht im Eingabebereich mit der linken Maustaste ein Anfragerechteck auf. Dies soll in jede Richtung möglich sein. Während der Ziehbewegung soll außerdem stets das aktuelle Rechteck angezeigt werden. Nur so wird dem Benutzer ein dynamischer Aufbau des Anfragerechtecks ermöglicht.

Nachdem die linke Maustaste losgelassen wurde, erfolgt eine Überprüfung des aufgezogenen Bereichs. Ein zu kleiner Abstand in x- oder y-Richtung deutet auf eine fehlerhafte Mausaktion hin. Es führt zu keiner Bereichsanfrage. Ist der Bereich hinreichend groß gewählt worden, so werden die Bildschirmkoordinaten von Anfangs- und Endpunkt in Weltkoordinaten umgerechnet.

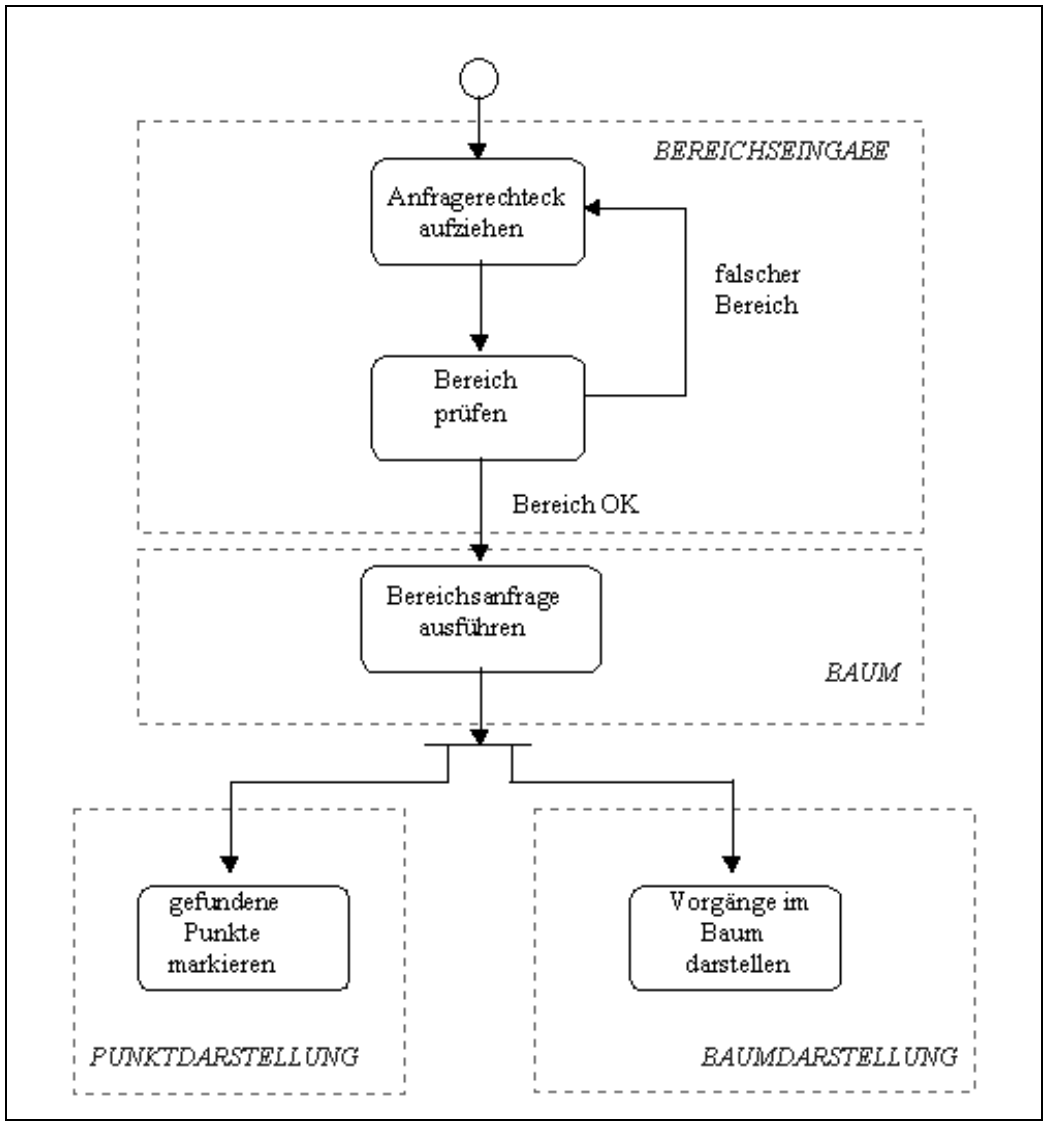


Abbildung 28: **Aktivitätsdiagramm „Anfragerechteck"**[58]

Mit diesen transformierten Werten führt das Programm im aktuellen 2-d-Baum eine Bereichsanfrage durch.

Die Überlegungen führen zu dem Aktivitätsdiagramm in Abbildung 28.

[58] Zur Erläuterung der Symbole siehe Tabelle A9

Visualisieren der Bereichsanfrage

Als Visualisierung des Anfragealgorithmus wird ausschließlich die farbliche Kennzeichnung der angefassten Knoten und der ausgewählten Blätter verwendet. Dies erscheint uns ausreichend, da im Eingabebereich gleichzeitig erkennbar ist, welche Punkte im Anfragerechteck liegen und welche Splitbereiche von dem Anfragerechteck geschnitten werden. Wie Abbildung 29 zeigt, werden die dabei besuchten Pfade und die gefundenen Punkte grün hervorgehoben.

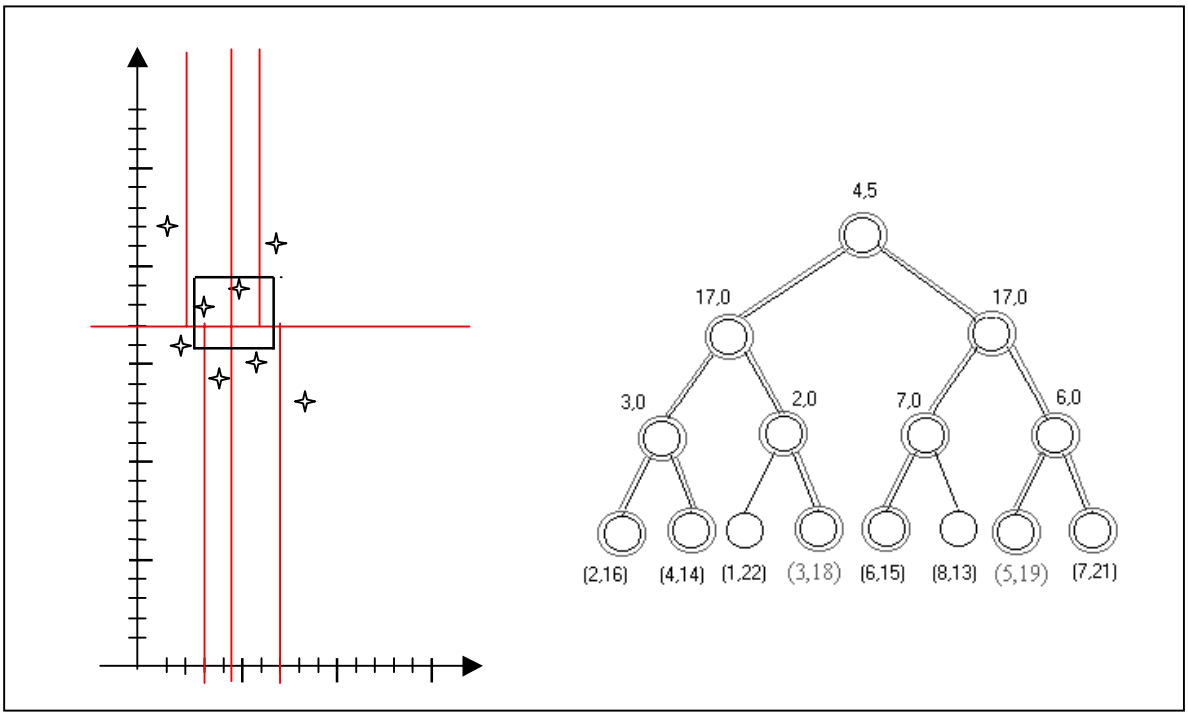


Abbildung 29: Beispielskizze zum Anfragerechteck

7.4.3.5 Aufbau des allgemeinen 2-d-Baums

Mit dem allgemeinen 2-d-Baum liegt ein ternärer Suchbaum vor. Für diesen ist der Algorithmus für das schöner Zeichnen von Kapitel 8.2.2 nicht unmittelbar verwendbar. Damit verbleiben uns folgende Möglichkeiten:

- Der Algorithmus für das schöner Zeichnen muss auf das Zeichnen ternärer Bäume angepasst werden.

- Falls an einem Knoten ein 1-d-Baum mit Punkten auf der Splitgeraden vorkommt, wird dies am Knoten in blau farblich kenntlich gemacht. Der 1-d-Baum wird auf Anforderung in einem separaten Fenster dargestellt. Die Anforderung erfolgt durch Klicken auf den betreffenden Knoten.

Auch in diesem Fall muss von unserer Seite eine Design-Entscheidung getroffen werden. Zur Entscheidungsfindung wollen wir betrachten, wie viele Knoten jeweils maximal pro Ebene bei einem ternären 2-d-Baum gezeichnet werden müssten. Dies kann der Tabelle 1 entnommen werden. Danach wird die maximale Knotenanzahl gegenüber dem binären Baum schnell sehr groß. Dies würde dazu führen, dass bereits ab der Ebene 4 der Baum nicht mehr komplett im Ausgabefenster darstellbar ist. Auch sehen wir die Gefahr des Verlustes der Übersichtlichkeit gegeben.
Deshalb entscheiden wir uns dafür, die jeweiligen 1-d-Bäume in separaten Fenstern anzeigen zu lassen. Dies bedeutet allerdings, dass eine Mausaktion im Ausgabebereich implementiert werden muss. Diese hat zunächst auszuwerten, welcher Knoten angeklickt wurde, um dann den zugeordneten 1-d-Baum in einem neu zu öffnenden Fenster anzuzeigen.

Baumhöhe	**Maximal mögliche Anzahl an Knoten beim 2-d-Baum**	**Maximal mögliche Anzahl an Knoten beim ternären 2-d-Baum**
0	1	1
1	2	3
2	4	8
3	8	27
4	16	48
5	32	128

Tabelle 1: Maximale Knotenanzahl des allgemeinen 2-d-Baums

7.4.3.6 Demo aufrufen

Es werden Demos zum dynamischen und zum ausgeglichenen 2-d-Baum angeboten. Nutzt der Benutzer eines der Demos, wird ihm in einer Sequenz gezeigt, wie die Algorithmen zum Einfügen

und Löschen von Punkten ablaufen und wie dabei die Splitgeraden im Eingabebereich eingezeichnet werden. Abschließend werden ein Anfragerechteck im Eingabebereich eingezeichnet und alle angefassten bzw. im Rechteck enthaltenen Punkte im Baum markiert. Nach jeder Aktion soll für den Benutzer die Möglichkeit bestehen, das Demo über eine Schaltfläche zu beenden. Während ein Demo abläuft, muss dem Benutzer durch einen erklärenden Text im Eingabebereich verdeutlicht werden, ob ein Punkt gerade eingefügt oder gelöscht oder eine Anfrage durchgeführt wird.

7.4.3.7 Spezielles Menüsystem

Dem Benutzer sollen die verschiedenen Baumvarianten, die Einstelloptionen für die Darstellungsart des Baumes und die Auswahl der Demos über ein Menüsystem angeboten werden. Deshalb wird für den 2-d-Baum zunächst das Menüsystem aus Abbildung 30 geplant. Es berücksichtigt auch bereits Menüpunkte, die erst später bei der Erweiterung des Programms aufgegriffen werden sollen.
Die Bedeutung der einzelnen Menüpunkte ist in der Bedienungsanleitung im Kapitel 10 zu finden.

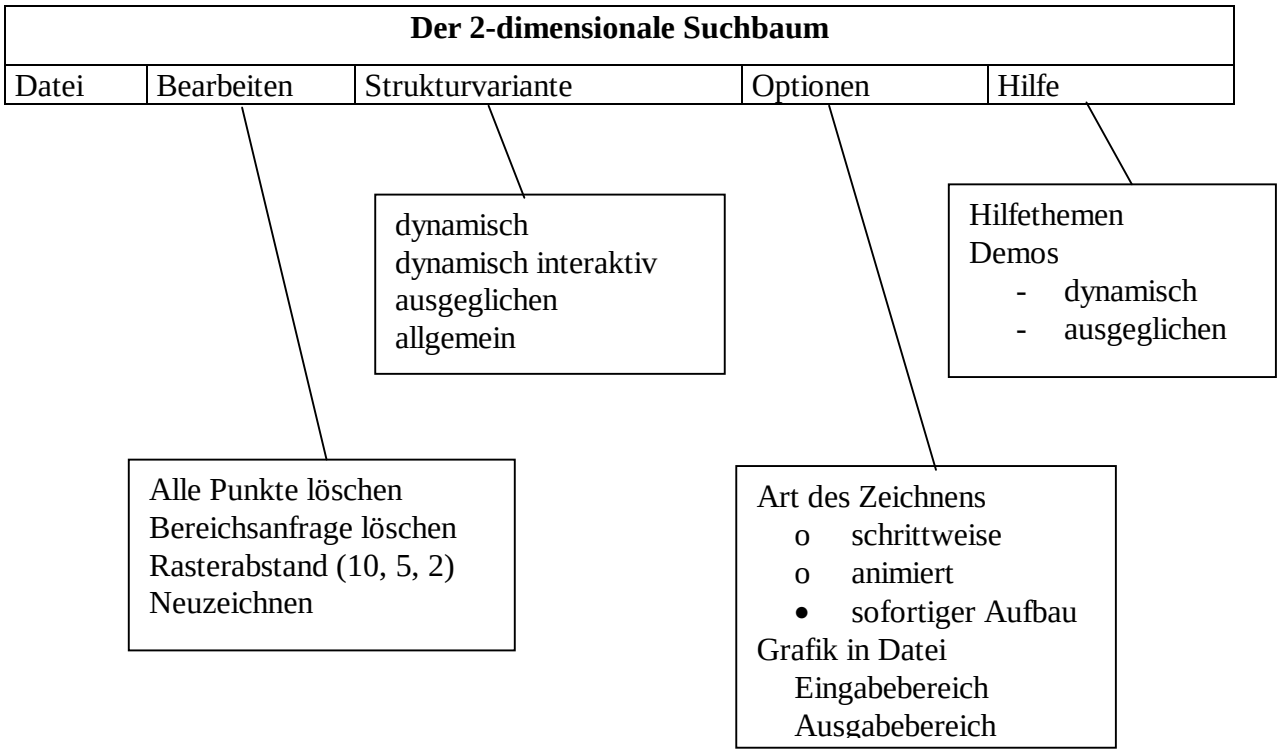


Abbildung 30: Menüsystem zum 2-d-Baum

7.5 Elaboration des Prioritätssuchbaums

7.5.1 Datenstruktur

Der Prioritätssuchbaum ist ein Blattsuchbaum für ein x-Intervall, in dem Punkte aus der Ebene nach folgenden Kriterien abgelegt werden:

- Jeder Punkt P=(x,y) steht auf dem Suchpfad zu seiner x-Koordinate.
- Längs eines Pfades von der Wurzel zu einem Blatt nehmen die y-Koordinaten der Punkte monoton zu. Die y-Werte im Baum bilden einen Minimumheap.
- Die Punkte stehen so dicht bei der Wurzel wie möglich.
- Zu Lernzwecken besitzen die Punkte zunächst paarweise verschiedene x-Koordinaten. Später wird auf eine Verarbeitung einer allgemeinen Punktmenge erweitert.

Der Prioritätssuchbaum lässt sich in unterschiedlichen Varianten realisieren, je nachdem wie viele Kenntnisse über die abzuspeichernden Punkte vorliegen:

- Ist die zu betrachtende Punktmenge bereits gegeben, so kann ein ausgewogener Gerüstbaum aufgebaut werden. In diesen lassen sich dann die Punkte nach den y-Koordinaten sortiert einfügen. Dieser Baum wird als statische Variante des Prioritätssuchbaums bezeichnet.
- Ist das Universum bekannt, aus dem die Punkte stammen können, wird zunächst ein Gerüstbaum für dieses Universum erstellt. Beim Auftreten der Punkte werden diese dynamisch in den Baum eingefügt. Dieser Baum wird als semidynamische Variante bezeichnet.
- Fehlt die Kenntnis über das Universum, muss der Baum völlig dynamisch aufgebaut werden. Pro auftretendem Punkt muss der zugrunde liegende Gerüstbaum zunächst um einen Knoten (x-Koordinate) erweitert werden, bevor dann der Punkt in diesen erweiterten Baum eingefügt werden kann.
- Beim dynamischen Baumaufbau besteht die Gefahr der Entartung. Es liegt schnell kein ausgewogener Skelettbaum mehr vor. Die Laufzeiten beim Einfügen, Löschen und der Bereichsanfrage verschlechtern sich entsprechend.

Deshalb wird als weitere Variante eine balancierte Version des Prioritätssuchbaums betrachtet, die nach den Kriterien für AVL-Bäume balanciert werden soll, da den Studenten diese Balancierungsform aus dem Grundstudium[59] bekannt ist.

- Bei der Betrachtung einer allgemeinen Punktmenge sind auch Punkte mit identischen x-Koordinaten zugelassen. Diese werden dann als zweidimensionales Datenobjekt ((x,y),y) betrachtet und zusätzlich die lexikographische Ordnung für die erste Komponente verwendet. Punkte, die im Baum keinen Platz finden, werden im Blattknoten in einer gleichermaßen sortierten Liste abgelegt.

7.5.1.1 Der statische Prioritätssuchbaum

Die Menge der zu speichernden Punkte wird in einer Liste nach den x-Koordinaten aufsteigend sortiert. Mit Hilfe dieser Liste erfolgt der Aufbau eines linksvollständigen Blattsuchbaums für die x-Koordinaten der Punkte.
In diesen Skelettbaum werden die Punkte nach den Kriterien des Prioritätssuchbaumes eingefügt. Dazu sind die Punkte nach ihren y-Koordinaten aufsteigend zu sortieren und in dieser Reihenfolge in das Baumgerüst einzufügen. Wegen der Sortierung der Punkte kann es dabei nicht zu einem Verdrängen bereits eingefügter Punkte kommen.

7.5.1.2 Der semidynamische Prioritätssuchbaum

Für die semidynamische Variante wird der Blattsuchbaum für das zugrunde liegende Universum erstellt. Dazu erfolgt das Abdecken des Bereichs zwischen der kleinsten und der größten x-Koordinate im ganzzahligen Bereich.
Auftretende Punkte werden dynamisch in den Blattsuchbaum eingefügt. Um den y-Heap zu erhalten, kann es dabei zum Verdrängen bereits vorhandener Punkte kommen.

[59] Hierbei handelt es sich um den Kurs Datenstrukturen, Kursnummer 01663, an der Fernuniversität Hagen.

Hinweis: Vereinfachend soll als Universum das mit 1 beginnende x-Intervall für die Skelettbäume betrachtet werden. Deshalb wird nur nach der oberen Grenze auf der x-Achse gefragt: 8, 16, 32 oder beliebig (aber maximal 32).

7.5.1.3 Der dynamische Prioritätssuchbaum

Der Einsatz „maßgeschneiderter" Techniken für das Einfügen und Löschen erlaubt auch eine effiziente dynamische Variante des Prioritätssuchbaumes[60].
Wir entscheiden uns für die volldynamische Variante aus [OTTM93] mit einem von der Reihenfolge der Punkte abhängigen natürlichen Blattsuchbaum als Suchstruktur:

- Der anfangs leere Baum besteht aus einem Knoten mit einem fiktiven Splitwert ∞, der größer als alle möglichen x-Koordinaten ist.
- Ein neuer Punkt wird in zwei Phasen eingefügt.
  Zunächst wird die x-Koordinate des Punktes verwendet, um den Blattsuchbaum um den entsprechenden Knoten zu erweitern.
  In den erweiterten Baum wird anschließend der Punkt wie üblich eingefügt.

7.5.1.4 Der balancierte dynamische Prioritätssuchbaum

Die Effizienz der dynamischen Variante ist nur gewährleistet, wenn der Baum ausgeglichen bleibt. Deshalb betrachten wir abschließend die balancierte dynamische Variante. Bei dieser erfolgt das Einfügen wie bei der dynamischen. Ist das AVL-Kriterium verletzt, so wird der zugrunde liegende Baum nach dem Einfügen eines neuen Punktes balanciert.
Beim Balancieren ist besonders darauf zu achten, dass der y-Heap nicht verletzt wird. Es sind bei jeder Rotation entsprechende Vorkehrungen zu treffen.

[60] Siehe Kapitel 7.5.1.4

7.5.2 Use Cases zum Prioritätssuchbaum

Nach den Überlegungen zur darzustellenden Datenstruktur werden jetzt exemplarische Anwendungsfälle zur groben Abstimmung dessen erarbeitet, was das zukünftige Programm für das Arbeiten mit dem Prioritätssuchbaum anbieten soll.
Der Benutzer soll zu verschiedenen Varianten des Prioritätssuchbaums Punkte einfügen und entfernen und eine Bereichsanfrage durch Festlegen eines Anfragehalbstreifens starten können. Nach allen Aktionen erfolgt ein Neuzeichnen des aktuellen Prioritätssuchbaums. Einführende animierte Demonstrationen sollen dem Benutzer das Arbeiten erleichtern. Dies führt zu dem allgemeinen Use-Case – Diagramm von Abbildung 31.

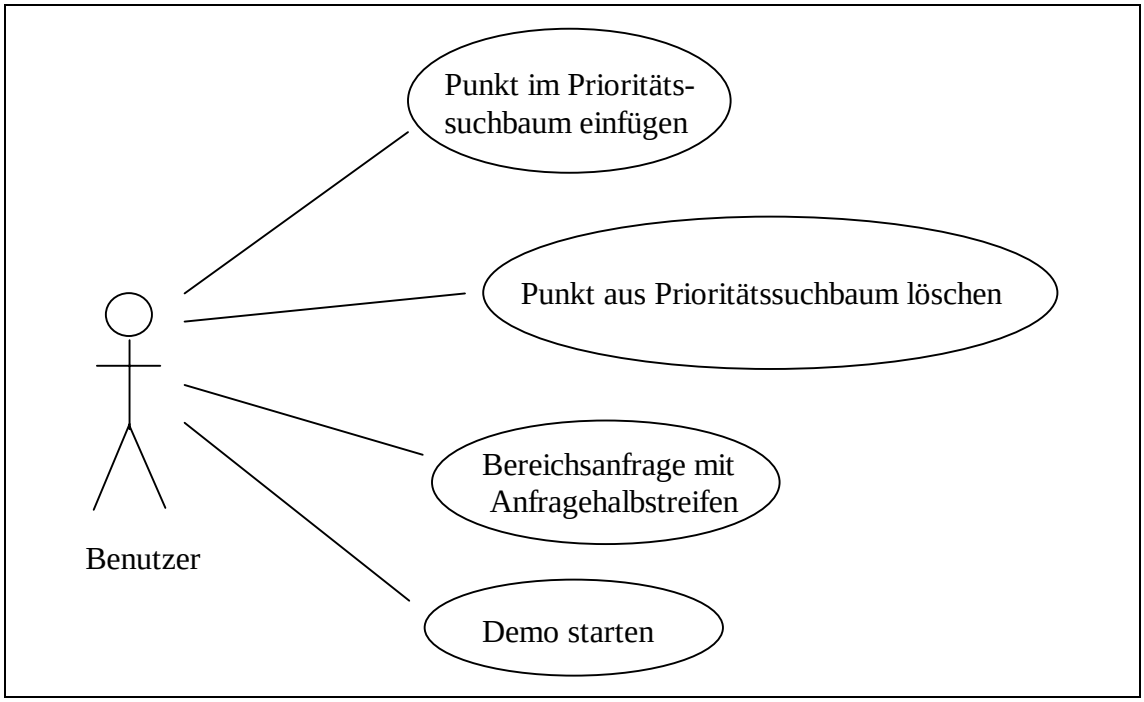


Abbildung 31: Use Case Diagramm[61]

Für die vier verschiedenen Arten des Prioritätssuchbaums lassen sich die Fälle „Punkt in Prioritätssuchbaum einfügen“ und „Punkt aus Prioritätssuchbaum löschen“ in Unterfälle aufteilen. Nachfolgend werden beispielhaft diese Anwendungsfälle zum dynamischen Prioritätssuchbaum aufgeführt.

[61] Zur Erläuterung der Symbole siehe Tabelle A8

Use Case „Punkt in den dynamischen Prioritätssuchbaum einfügen"

- Der Benutzer klickt einen Punkt in der Ebene (1. Quadrant) mit der Maus an.
- Der Punkt wird in den Eingabebereich eingezeichnet.
- Ein neuer Blattknoten mit der x-Koordinate des Punktes wird im Baum ergänzt.
- Der Punkt wird in den Prioritätssuchbaum eingefügt.
- Im Ausgabebereich wird der Baum auf unterschiedliche Weise gezeichnet:
  - Der erweiterte Baum mit dem neu eingefügten Punkt wird sofort angezeigt.
  - Die Abläufe beim Einfügen des neuen Punktes werden schrittweise in einer Animation verdeutlicht.

Use Case „Punkt aus dem dynamischem Prioritätssuchbaum löschen"

- Der Benutzer wählt mit der Maus einen vorhandenen Punkt zum Löschen aus.
- Der Punkt wird in der Ebene (1. Quadrant) gelöscht.
- Der Punkt wird aus dem Prioritätssuchbaum entfernt.
- Die im y-Heap entstandene Lücke wird geschlossen.
- Zusätzlich werden die Knoten (Blatt und innerer Knoten) mit der x-Koordinate des gelöschten Punktes aus dem Baum entfernt, ohne dabei den y-Heap zu verletzen.
- Der aktualisierte Baum wird neu angezeigt.

Use Case „Halbstreifenbereichsanfrage durchführen"

- Im Eingabebereich zieht der Benutzer mit der Maus den Bereich für die Anfrage (Halbstreifen) auf.
- Im Baum wird das markierte x-Intervall durchsucht, jedoch nur solange die y-Werte der gespeicherten Punkte nicht größer als der obere y-Grenzwert des Halbstreifens sind.
- Die bei der Suche aufgefundenen Punkte und die besuchten Knoten werden farblich hervorgehoben in den Prioritätssuchbaum eingezeichnet.

Use Case „Demo aufrufen"

- Der Benutzer wählt im Menü Hilfe eines der angebotenen Demos aus.
- Das Demo zeigt in einem „Film" die Vorgänge beim Einfügen und Löschen von Punkten und bei der Bereichsanfrage im Baum.
- Über eine Schaltfläche kann das Demo jederzeit vom Benutzer wohldefiniert abgebrochen werden, d. h. das Demo wird den aktuellen Arbeitsschritt abschließen, bevor es endet.

7.5.3 Verfeinerte Analyse mit Aktivitätsdiagrammen

Einen ersten Übergang von der Analyse

"Was soll gemacht werden?"

zum Entwurf

"Wie soll dies geschehen?"

bieten Aktivitätsdiagramme[62]. Sie stellen verfeinerte Abläufe der Anforderungen dar und bringen Möglichkeiten ins Spiel, wie diese Forderungen erfüllt werden können.

[62] Sinngemäß nach [OEST98]

Dies wird hier für die zuvor genannten Anwendungsfälle durchgeführt. Dabei erfolgen auch Überlegungen, wie die Visualisierung der Baumalgorithmen durchgeführt werden kann.

7.5.3.1 Punkt in dynamischen Prioritätssuchbaum einfügen

Der Benutzer klickt im Eingabebereich einen Punkt mit der Maus an. Die ermittelten Bildschirmkoordinaten müssen in Punkte des ersten Quadranten umgerechnet werden. Dabei ist das eingestellte Raster zu berücksichtigen. Die Punkte werden „eingefangen“ und zu Rasterpunkten umgerechnet.

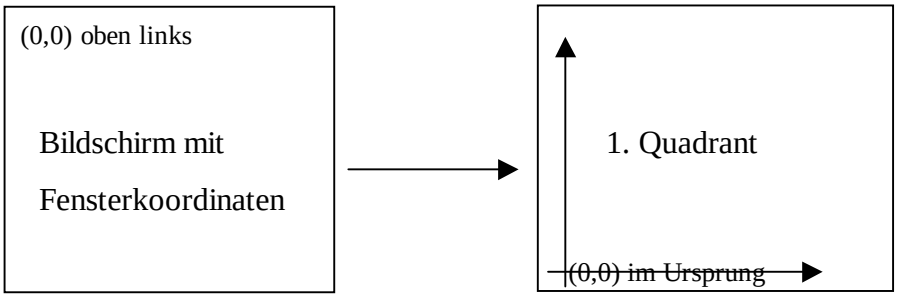


Abbildung 32: Koordinatentransformation

Bevor der ermittelte Punkt in den Prioritätssuchbaum eingefügt werden kann, muss geklärt werden, ob dieser Punkt im Rahmen der Einstellungen erlaubt ist:

- Kommt der Punkt bereits vor, so darf er kein zweites Mal eingefügt werden.
- Sind nur Punkte mit disjunkten x-Koordinaten erlaubt, so wird der neue Punkt nicht eingefügt, wenn bereits ein Punkt mit diesem x-Koordinatenwert vorkommt.
- Liegt der Punkt im semidynamischen Fall außerhalb des eingestellten Universums, muss er abgewiesen werden.

All diese Tests benötigen eine aktuelle Punktmenge. Es ist deshalb erforderlich, auch die Punktmenge in einer Datenstruktur festzuhalten. Da dem Benutzer die Wiederholungsmöglichkeit des Baumaufbaus angeboten werden soll, eignet sich dafür eine Liste, die die eingegebenen Punkte in der Eingabereihenfolge festhält.
Zum Testen wird auf diese Punktliste zugegriffen. Ist der Punkt zugelassen, wird er an die aktuelle Punktliste angehängt und in den Prioritätssuchbaum eingefügt.

Der akzeptierte neue Punkt wird im 1. Quadranten eingezeichnet. Der aktualisierte Baum wird in der Zeichenfläche neu angezeigt. Dabei müssen die vorgenommenen Veränderungen für den Benutzer deutlich werden (siehe Gedanken zum Visualisieren auf Seite 71).

Als Ergebnis dieser detaillierteren Überlegungen ergibt sich das Aktivitätsdiagramm von Abbildung 33 für das Einfügen eines Punktes in den dynamischen Prioritätssuchbaum. Es zeigt die ermittelten Aktivitäten und ihre Abfolge. Werden darin die Aktivitäten nach Zuständigkeiten gruppiert, so kommen die gestrichelt eingezeichneten Komponenten zustande, die sich als Kandidaten für eine grobe Klasseneinteilung anbieten.

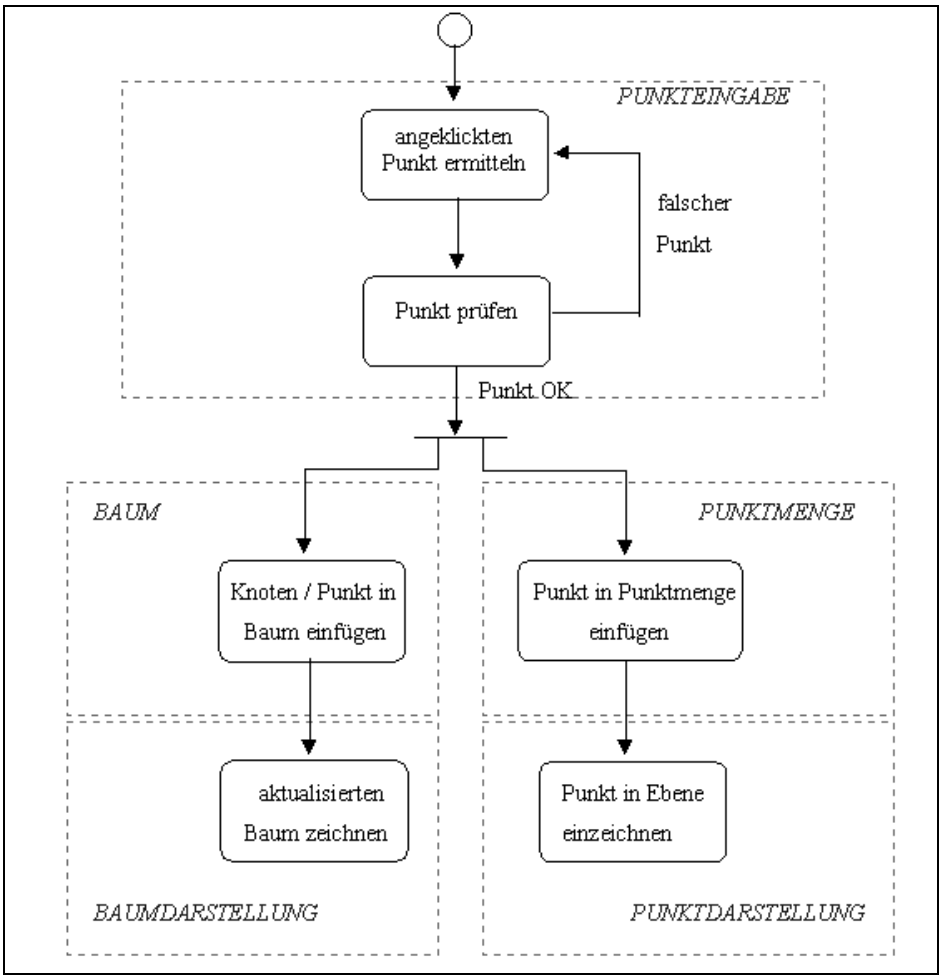


Abbildung 33: Aktivitätsdiagramm „Punkt in Prioritätssuchbaum einfügen"[63]

63 Zur Erläuterung der Symbole siehe Tabelle A9

Überlegungen zum Visualisieren

Wünschenswert ist eine animierte schrittweise Darstellung aller Abläufe in der Baumzeichenfläche. Zunächst jedoch soll zum Testen der Baumalgorithmen die Visualisierung in einfacher Form vorgenommen werden. Der Baum wird nach jeder Veränderung sofort in seiner neuen Form angezeigt. Knoten und Kanten werden schwarz gezeichnet, die Splitwerte an den Knoten ebenfalls schwarz eingetragen. Die Punkte im y-Heap werden blau dargestellt.

Dieser sofortige Zeichenmodus soll auch später als Option im Programm erhalten bleiben, weil er ein schnelleres Arbeiten erlaubt, wenn der Benutzer die Algorithmen erst einmal verstanden hat. Im folgenden werden deshalb stets Überlegungen zum Sofortmodus und zur animierten Darstellung vorgenommen.

7.5.3.2 Visualisieren des Einfügevorgangs

Sofortmodus:

- Semidynamisch-statische Variante:
  Erst nach der Eingabe aller Punkte wird daraus der komplette Baum aufgebaut. Er wird dabei ohne Farbmarkierungen gearbeitet, weil keinerlei dynamische Veränderungen aufzuzeigen sind.

- Semidynamische Variante:
  Der Einfügepfad zum Blatt mit der x-Koordinate des neuen Punktes wird grün markiert. Neu eingefügte Punkte werden durch rote Farbe deutlich gemacht. Dies gilt auch für verdrängte Punkte.
  So ist sofort für den Benutzer ersichtlich, wo der neue Punkt eingefügt wurde. Sind mehrere Punkte rot, lässt sich daran der Pfad verfolgen, auf dem früher eingefügte Punkte durch den neuen verdrängt wurden.

- Dynamische / dynamisch balancierte Variante:
  Auch der neu eingefügte Blattknoten wird durch rote Farbe hervorgehoben.
  Für die Punkte gilt gleiches wie im semidynamischen Fall.

Animation:

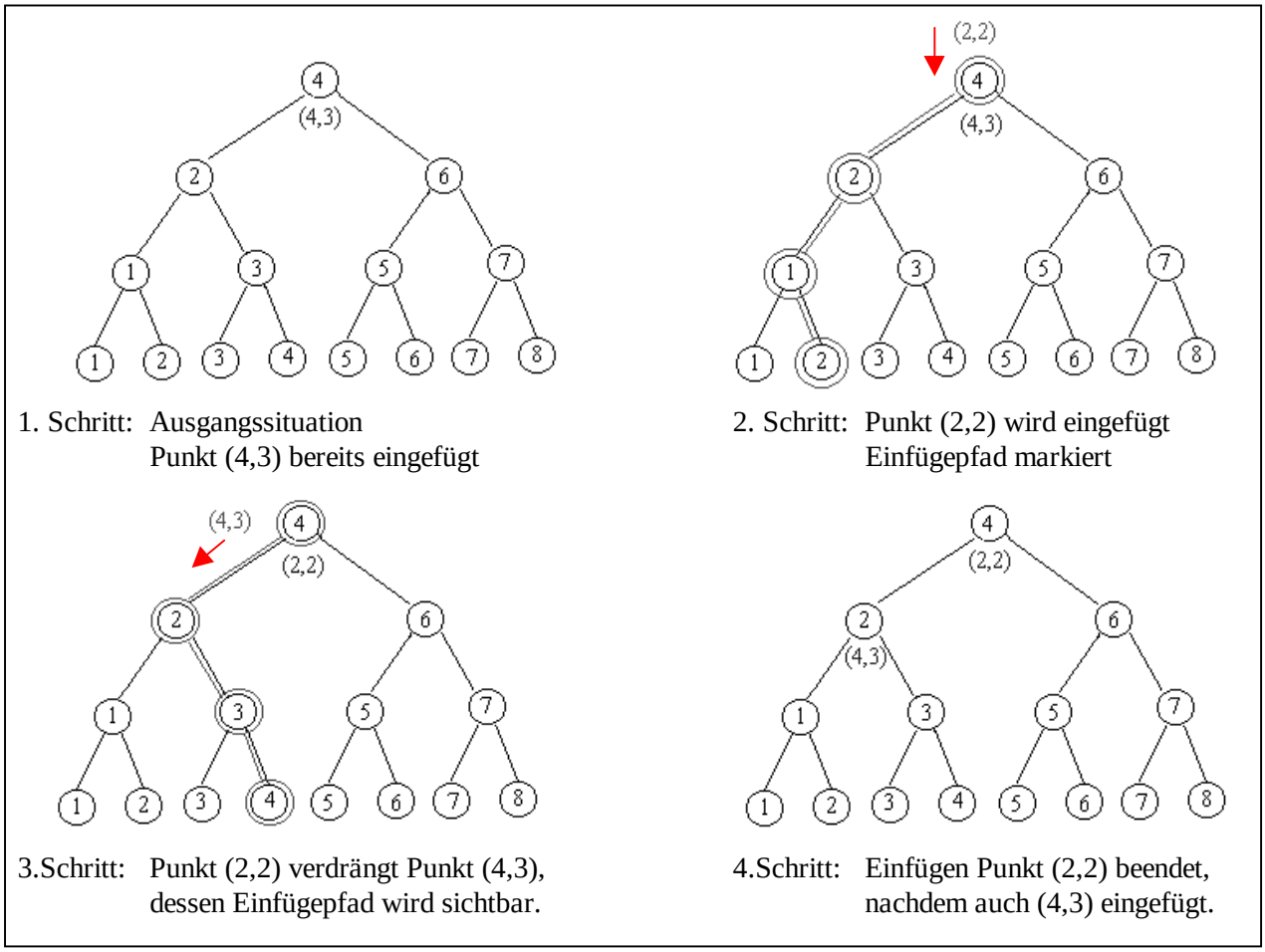
- Semidynamisch-statische Variante:

  Der Skelettbaum wird aufgebaut und ohne besondere Hervorhebungen angezeigt. Der Einfügepfad wird wie im Sofortmodus grün markiert.

  Anschließend wandert Punkt für Punkt in roter Farbe (neuer Punkt) auf dem Einfügepfad langsam an seine Position im Baum und wird dort in blauer Farbe abgelegt.

- Semidynamisch:

  Der Skelettbaum ist ständig eingezeichnet (Abbildung 34). Ein neuer Punkt wandert in rot auf dem Weg zum Blatt mit seinem x-Koordinatenwert von der Wurzel aus in den Baum hinein. Trifft der neue Punkt im Knoten auf einen Punkt mit größerem y-Wert als er selbst aufweist, so verdrängt er diesen Punkt, d. h. der neue Punkt wird blau in den Knoten eingefügt und der verdrängte Punkt wandert dafür in Rot weiter in den Baum hinein, jetzt jedoch auf dem Pfad zu seinem x-Koordinatenwert. Abbildung 34 zeigt das Einfügen des Punktes (2|2).



1. Schritt: Ausgangssituation
Punkt (4,3) bereits eingefügt

2. Schritt: Punkt (2,2) wird eingefügt
Einfügepfad markiert

3.Schritt: Punkt (2,2) verdrängt Punkt (4,3), dessen Einfügepfad wird sichtbar.

4.Schritt: Einfügen Punkt (2,2) beendet, nachdem auch (4,3) eingefügt.

Abbildung 34: Animiertes Einfügen eines Punktes

- Dynamisch / dynamisch balanciert:

  Beim dynamischen Prioritätssuchbaum unterscheidet sich das eigentliche Einfügen der Punkte nicht von den Vorgängen im semidynamischen Fall.

  Hier wird jedoch der zugrunde liegende Skelettbaum erst um ein Blatt mit dem neuen x-Koordinatenwert erweitert. Dazu muss zunächst der Ansatzknoten für das neue Blatt gesucht werden. In roter Farbe wird gezeigt, wie der Suchpfad langsam wächst, bis der Ansatzknoten erreicht ist.

  Dann soll zuerst der linke Sohn mit dem kleineren x-Wert aus dem Ansatzknoten „herausfallen", anschließend der rechte Sohn. Der Punkt wird abschließend in den erweiterten Baum eingefügt. Abbildung 35 zeigt das Einfügen der Punkte (5|2) und (2|4) in den zunächst leeren Baum.

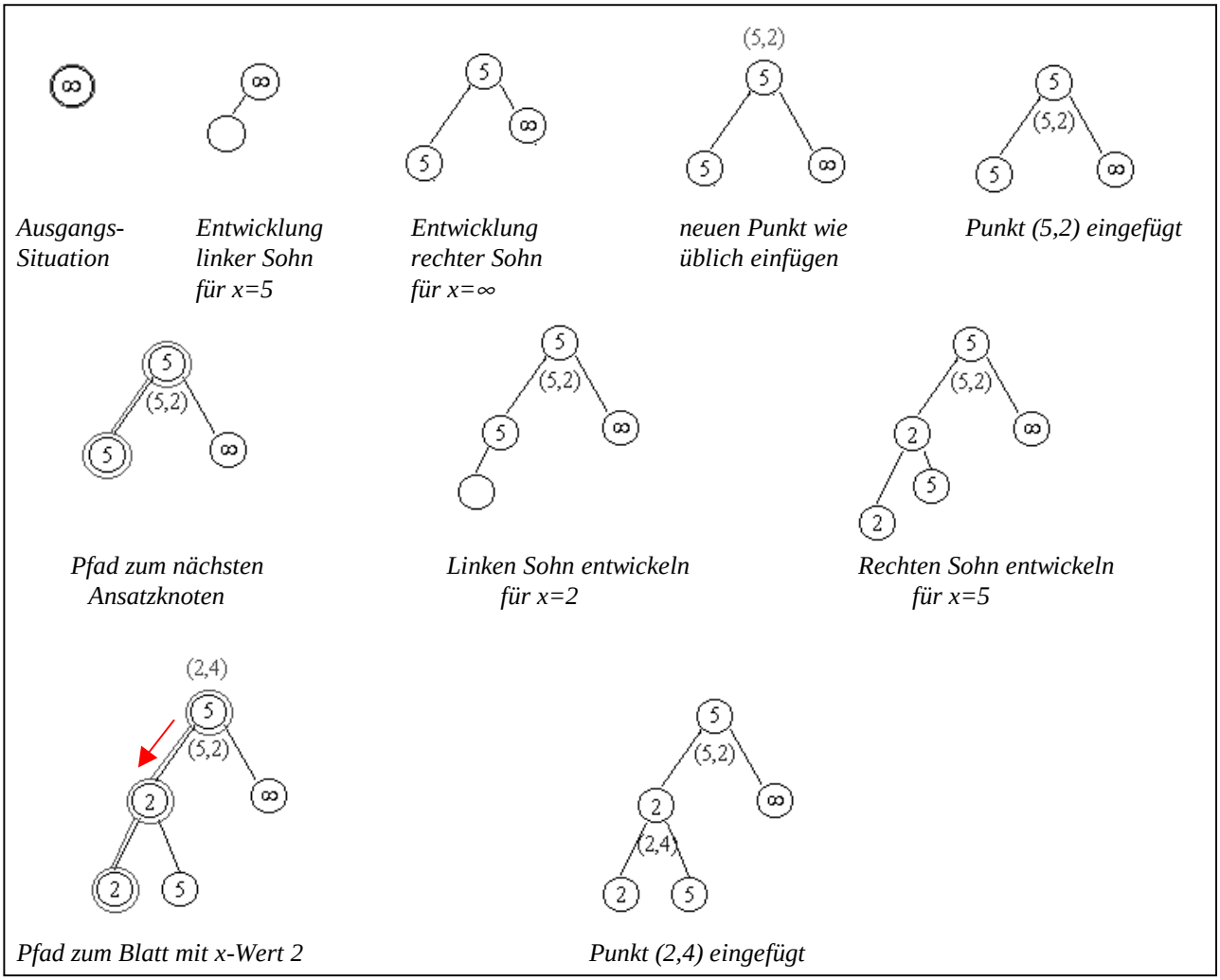

Ausgangs-Situation
Entwicklung linker Sohn für x=5
Entwicklung rechter Sohn für x=∞
neuen Punkt wie üblich einfügen
Punkt (5,2) eingefügt
Pfad zum nächsten Ansatzknoten
Linken Sohn entwickeln für x=2
Rechten Sohn entwickeln für x=5
Pfad zum Blatt mit x-Wert 2
Punkt (2,4) eingefügt

Abbildung 35: Beispielskizzen zum animierten Einfügen in den dynamischen Baum

Wenn der Baum beim dynamischen Aufbau das AVL-Kriterium verletzt, muss er balanciert werden.

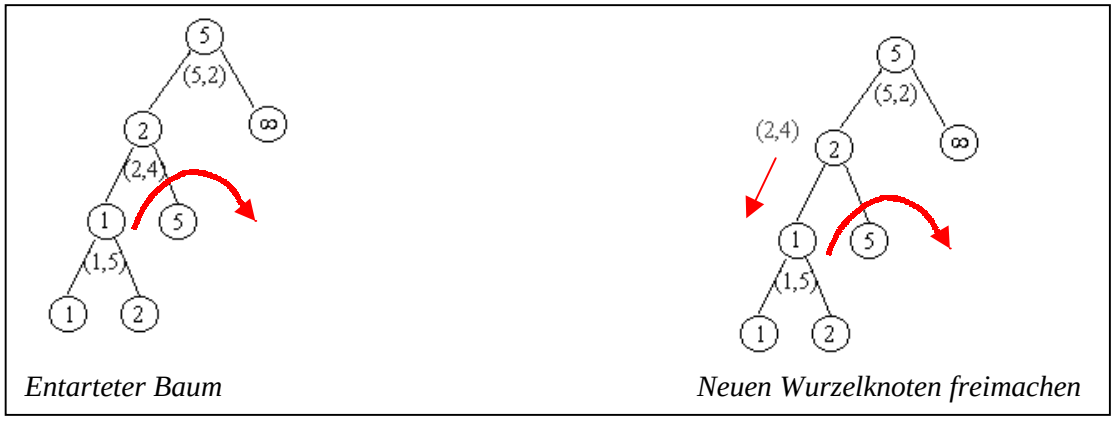


Abbildung 36: Balancieren im Prioritätssuchbaum (Teil 1)

Wird der Baum aus Abbildung 35 um den Punkt (1|5) erweitert, ergibt sich zunächst der entartete Baum aus Abbildung 36. Im Wurzelknoten ist das AVL-Kriterium verletzt. Der Baum muss durch eine einfache Rotation nach rechts balanciert werden. Vorher ist der Punkt (2|4) in den richtigen Teilbaum (hier den linken) nach dem üblichen Algorithmus einzufügen, um den neuen Wurzelknoten frei zu machen.

Abbildung 37 verdeutlicht, dass der Punkt aus dem alten Wurzelknoten (5) in den neuen Wurzelknoten (2) übernommen wird. Dann kann die eigentliche Rotation durchgeführt werden, ohne den y-Heap zu verletzen.

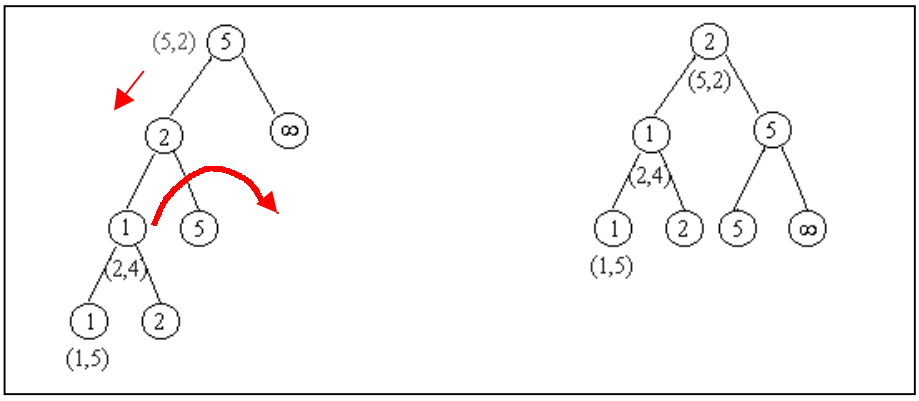


Abbildung 37: Balancieren im Prioritätssuchbaum (Teil 2)

Bei einer Doppelrotation werden die beschriebenen Abläufe für jede einzelne Rotation benötigt.

7.5.3.3 Punkt aus dem dynamischen Prioritätssuchbaum löschen

Die detaillierteren Ausführungen zum Löschen eines Punktes unterscheiden sich vom Einfügen eines Punktes nur unwesentlich.
In der Eingabefläche klickt der Benutzer den zu löschenden Punkt mit der rechten Maustaste an. Auch in diesem Fall muss der Punkt überprüft werden. Kommt er in der aktuellen Punktliste vor, so muss er sowohl aus dem Baum als auch aus der Punktmenge gelöscht werden. Der aktualisierte Baum und die neue Punktmenge sind am Bildschirm neu darzustellen.
Da sich das Aktivitätsdiagramm zum Löschen nur durch die verwendete Methode vom Einfügen unterscheidet, wird hier auf seine Darstellung verzichtet.

7.5.3.4 Visualisieren des Löschvorgangs

Wie beim Einfügen (siehe Seite 71) erfolgt die Unterscheidung zwischen **Sofortmodus** und **Animation**.

Sofortmodus:

- Semidynamisch-statische Variante:
  Nach jeder Veränderung an der aktuellen Punktmenge muss der Benutzer den Baum komplett neu aufbauen und zeichnen lassen. Deshalb werden hier keine farblichen Hervorhebungen benötigt.
- Semidynamische Variante:
  Beim Löschen werden die Punkte rot hervorgehoben, die nach dem Löschen eines Punktes „hochgezogen“ wurden, um die entstandenen Lücken zu schließen. Durch den Farbeinsatz lässt sich im Nachhinein gut erkennen, auf welchem Pfad dies erfolgt ist.
- Dynamische / dynamisch balancierte Variante:
  Die entfernten Knoten werden im aktualisierten Baum nicht mehr angezeigt. Für die Punkte gilt gleiches wie im semidynamischen Fall. Der Löschvorgang kann im dynamischen Fall aber auch einen Einfügevorgang enthalten (Punkt aus Knoten verdrängen, bevor dieser gelöscht werden darf). Weil alle Punkte mit einer neuen Position im Baum eingefärbt werden, können deshalb an verschiedenen Stellen rote Markierungen auftreten.

Animation:

- Semidynamisch-statische Variante:

  Es gibt keine Löschdynamik. Der Baum wird, wie beim animierten Punkteinfügen auf Seite 72 beschrieben, nach dem Löschen von Punkten komplett neu aufgebaut.

- Semidynamisch-statische Variante:

  Der Skelettbaum wird ständig unverändert angezeigt, nur das Löschen der Punkte muss animiert werden.

  Zunächst wird dem Betrachter das Auffinden des Punktes gezeigt, indem der Suchpfad in roter Farbe eingezeichnet wird. Der gefundene Punkt wird vom Bildschirm gelöscht und der Punkt, der die Lücke füllen soll, „läuft“ von seinem Knoten „hoch” zum jetzt leeren Knoten. Die letzte Aktion wiederholt sich, bis alle dabei entstehenden Lücken geschlossen sind.

  Abbildung 38 zeigt das Löschen des Punktes (2|4).

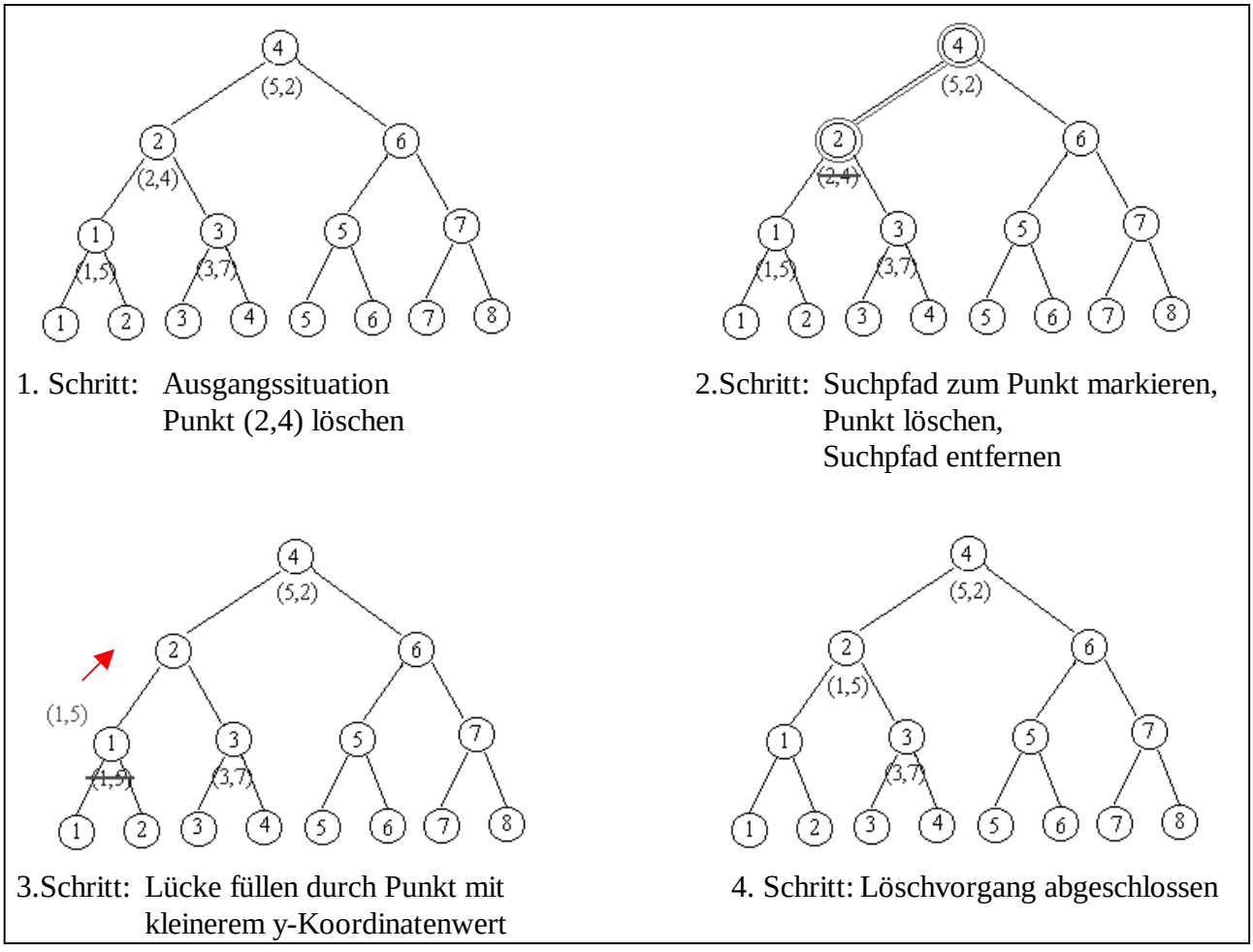


Abbildung 38: Animiertes Löschen im semidynamischen Prioritätssuchbaum

- Dynamische / dynamisch balancierte Variante:

  Beim dynamischen Prioritätssuchbaum unterscheidet sich das eigentliche Löschen der Punkte nicht von den Vorgängen im semidynamischen Fall.

  Hier müssen jedoch zusätzlich im zugrunde liegenden Skelettbaum die Knoten mit dem x-Koordinatenwert des gelöschten Punktes entfernt werden:

  - ♦ Zunächst wird der Suchpfad bis zum Vaterknoten des zu löschenden Blattes markiert.
  - ♦ Ein im markierten Endknoten vorhandener Punkt wird in den verbleibenden Teilbaum verlagert.
  - ♦ In rot wird die neue Kante vom Vorgänger des markierten Endknotens zum verbleibenden Nachfolger eingezeichnet. Das umgangene Knotenpaar ist zusammen mit den Splitwerten und den alten Kanten gelöscht. Der Splitwert im inneren "Löschknoten" wird mit dem Nachfolgewert überschrieben.
  - ♦ Nach kurzer Pause ist der Ergebnisbaum mit korrigierten Knotenpositionen zu zeichnen.

  Abbildung 39 zeigt das Löschen des Punktes (2|4) im Baum aus Abbildung 37.

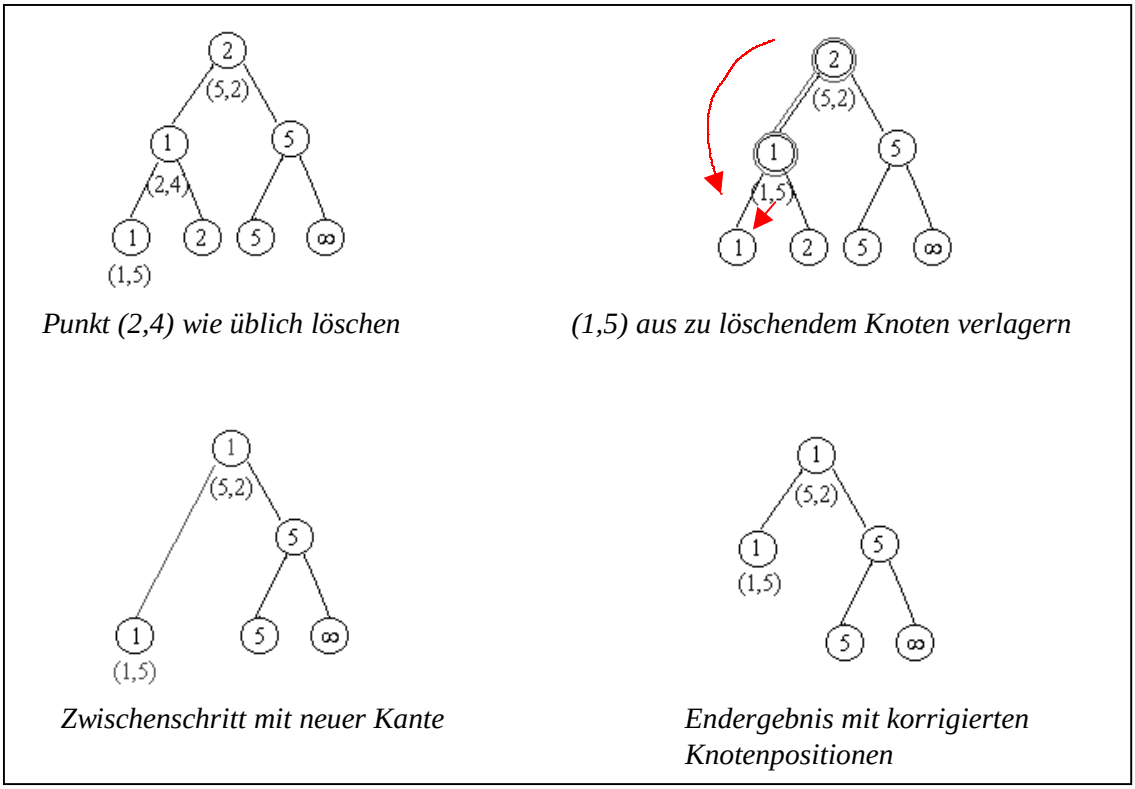


Abbildung 39: Animiertes Löschen im dynamischen Prioritätssuchbaum

7.5.3.5 Bereichsanfrage durchführen

Der Benutzer zieht im Eingabebereich mit der Maus einen nach unten offenen Halbstreifen auf. Dies soll in jede Richtung möglich sein. Während der Ziehbewegung wird stets der aktuelle Halbstreifen angezeigt, damit er dynamisch korrigiert werden kann.
Beim Loslassen der Maustaste werden die aktuellen Werte für das x-Intervall und den y-Grenzwert ermittelt. Der aufgezogene Bereich muss überprüft werden. Ein zu kleiner Abstand in x-Richtung deutet auf eine fehlerhafte Mausaktion hin und darf zu keiner Bereichsanfrage führen.
Ist der Bereich erfolgreich überprüft, werden die Werte für das x-Intervall und der y-Grenzwert in Werte für den 1. Quadranten umgerechnet.

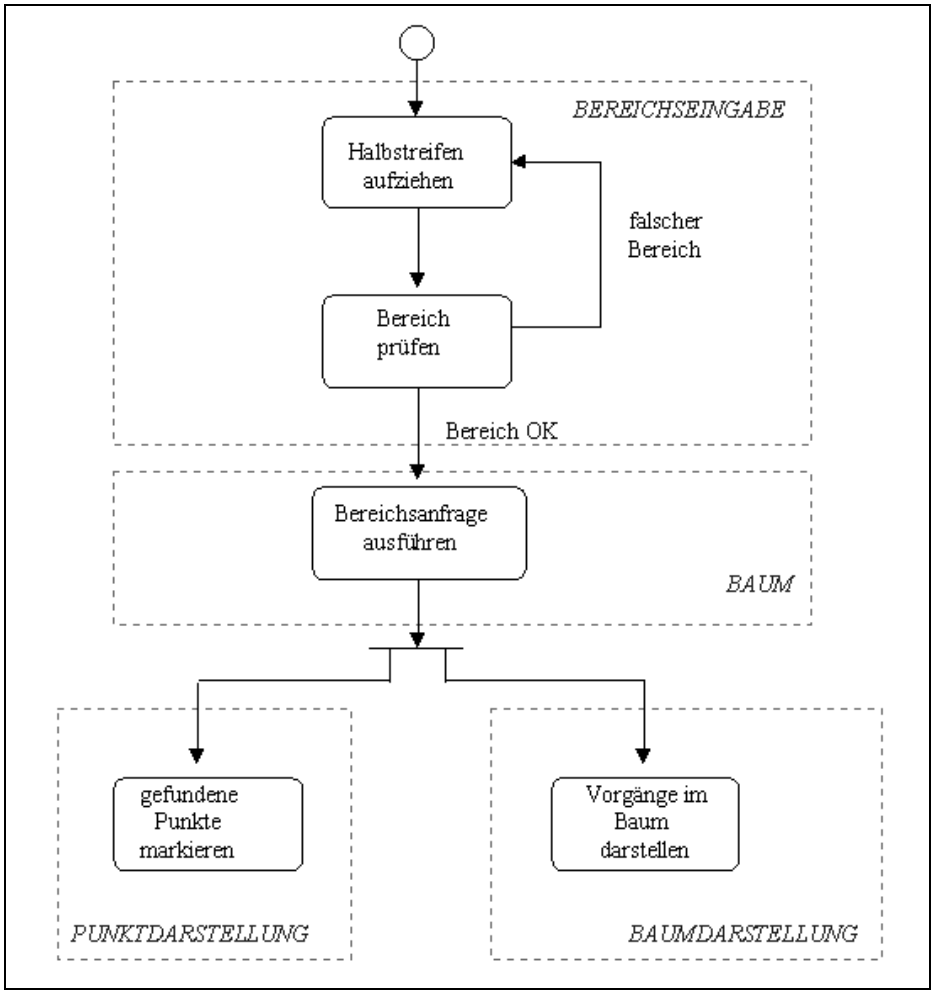


Abbildung 40: **Aktivitätsdiagramm „Bereichsanfrage"**[64]

[64] Zur Erläuterung der Symbole siehe Tabelle A9

Mit den transformierten Werten führt das Programm im aktuellen Prioritätssuchbaum eine Bereichsanfrage durch. Die dabei durchlaufenen Pfade werden in der Zeichenfläche illustriert.
Die während der Anfrage gefundenen Punkte werden im Baum und im 1. Quadranten markiert.
Diese Überlegungen führen zu dem Aktivitätsdiagramm von Abbildung 40. Auch in diesem Diagramm sind Zuständigkeiten eingezeichnet, um die spätere Klassenbildung zu unterstützen.

Visualisieren der Bereichsanfrage

Wie beim Einfügen von Punkten (siehe Seite 71) ist zwischen Sofortmodus und Animation zu unterscheiden.

Sofortmodus:

Für die Bereichsanfrage werden in allen Varianten die im Baum besuchten Pfade und die gefundenen Punkte (Ergebnis der Anfrage) grün markiert eingezeichnet.

Animation:

Die Bereichsanfrage wird für alle Varianten identisch durchgeführt (siehe Abbildung 41).
Das Besuchen der Pfade im Baum wird in Besuchsreihenfolge kontinuierlich in grüner Farbe nachgezeichnet, der Benutzer sieht den besuchten Bereich langsam wachsen. Liegt im aktuellen Knoten ein Punkt aus dem Anfragebereich, wird auch dieser grün eingefärbt. In der Eingabefläche wird der gefundene Punkt ebenfalls farbig hervorgehoben.

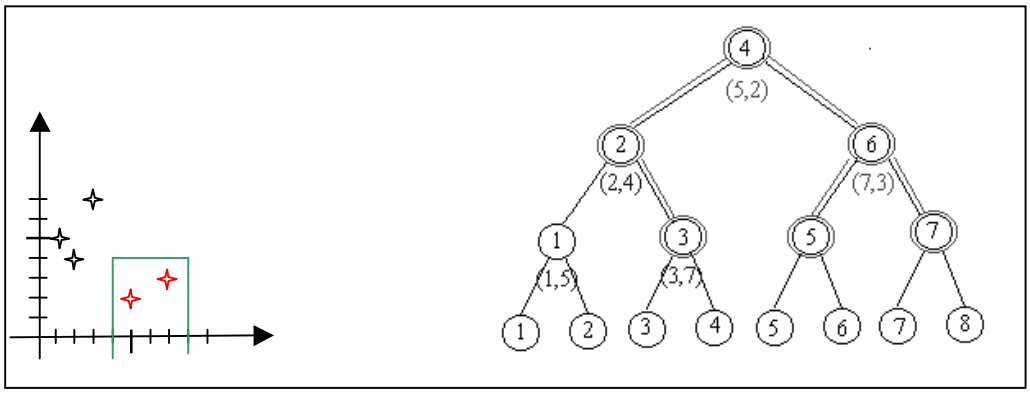


Abbildung 41: Beispielskizze zur Bereichsanfrage

7.5.3.5 Demo einsetzen

Es werden Demos zur semidynamischen und dynamischen Variante angeboten. Startet der Benutzer eines der Demos, wird ihm in einer Sequenz gezeigt, wie die Algorithmen zum Einfügen und Löschen von Punkten ablaufen. Abschließend wird eine Bereichsanfrage demonstriert. Nach jeder Aktion kann der Benutzer über eine Schaltfläche das Demo beenden.
In beiden Demos wird die gleiche Punktmenge verwendet, so dass der Benutzer die Unterschiede zwischen den Varianten gut beobachten kann. Die Demos greifen auf die animierten Darstellungen der verschiedenen Abläufe zurück, fassen diese in einem „Film" zusammen.
Während ein Demo abläuft, muss für den Benutzer durch erklärenden Text deutlich werden, welcher Vorgang gerade abläuft und welcher Punkt eingefügt oder gelöscht wird, da er nur passiver Betrachter ist und nicht selbst agiert.

7.5.3.7 Spezielles Menüsystem zum Prioritätssuchbaum

Dem Benutzer sollen die verschiedenen Baumvarianten, die Einstelloptionen für die Darstellungsart des Baumes sowie die Auswahl der Demos über ein Menüsystem angeboten werden. Deshalb wird zunächst das Menüsystem aus Abbildung 42 geplant. Es berücksichtigt auch bereits Menüpunkte, die erst später bei der Erweiterung des Programms aufgegriffen werden sollen.
Die Bedeutung der einzelnen Menüpunkte ist der Bedienungsanleitung im Kapitel 10 entnehmbar.

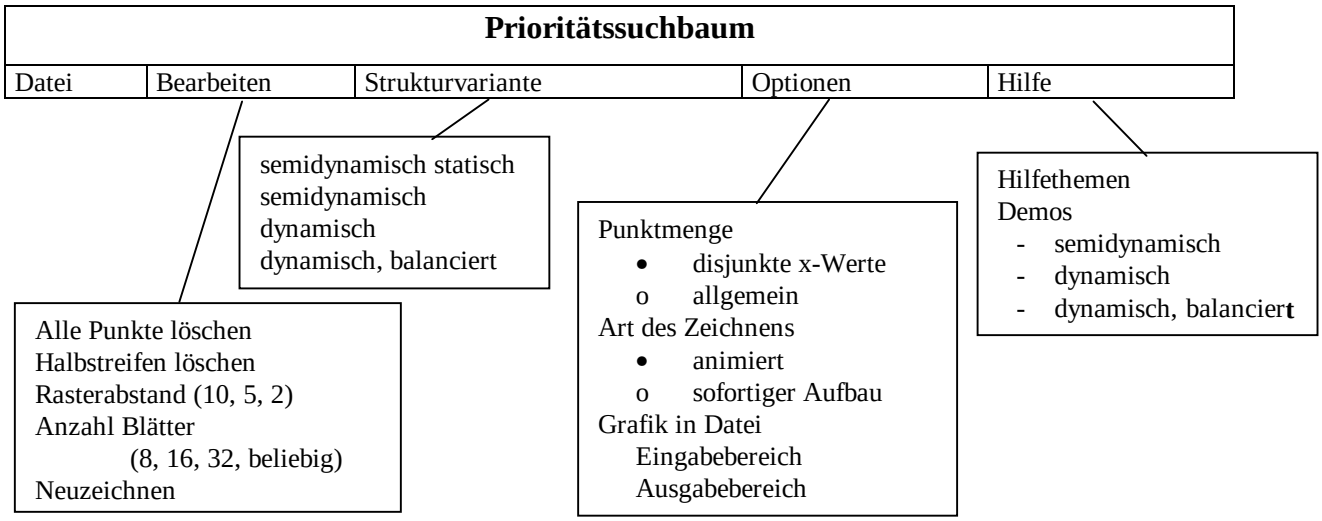


Abbildung 42: Spezielles Menüsystem zum Prioritätssuchbaum

7.6 Vorläufige Klassenbildung

Zur ersten groben Klassenbildung betrachten wir den Text der Use Cases aus den Kapiteln 7.4.2. und 7.5.2.
Der Begriff Baum steht dabei stellvertretend für den 2-d-Baum oder den Prioritätssuchbaum.

Die Substantive

Baum,	**Eingabebereich,**	**Demo,**
Knoten,	**Ausgabebereich,**	**Animation**
Punkt,	**Menü,**	

deuten auf mögliche Klassenkandidaten hin.

- Den **Baum** fassen wir als Menge (Behälter) von Baumknoten auf, deshalb entscheiden wir uns für eine Klasse **`Node`**[65] mit den Attributen des Knotens und eine Klasse **`Tree`**, die die Methoden bezüglich der Knotenmenge enthält, z. B. die typischen Behälteroperationen Einfügen, Löschen und Suchen eines Knotens.
  Die Aktivitäten aus den verfeinerten Use Cases zum Aktualisieren des Prioritätssuchbaumes
  `Punkt in Baum einfügen,`
  `Punkt aus Baum löschen,`
  `Bereichsanfrage im Baum`
  müssen offensichtlich der Klasse **`Tree`** als Methoden zugeordnet werden.

- Für die Klasse **Punkt** nutzen wir die fertige Java-eigene Klasse **`Point2D.Double`**.

- Die Klasse **Eingabebereich** (**`InputArea`**) muss Methoden enthalten für das
  `Zeichnen des 1. Quadranten,`
  `Zeichnen der Punktmenge,`
  `Reagieren auf MouseEvents,`
  `Reagieren auf MouseMotionEvents.`

[65] Zur besseren Lesbarkeit schreiben wir Klassennamen fett, Variablen kursiv und Methoden in Courier.

- Die Klasse **Ausgabebereich** muss eine Methode zum
  `Zeichnen des Baumes`
  enthalten. Wir bezeichnen sie deshalb als **`TreeCanvas`**.

Ein Blick auf die Verfeinerungen und Aktivitätsdiagramme in den Kapiteln 7.4.3 und 7.5.3 mit den Komponenten

BAUMDARSTELLUNG *PUNKTDARSTELLUNG* *BAUM*
PUNKTMENGE *BEREICHSEINGABE* *PUNKTEINGABE*
DARSTELLUNG DER EINGABEFLÄCHE

bestätigt unsere bisherigen Entscheidungen für die vorläufige Klassenbildung.

Die *Baumdarstellung* erfolgt durch die Zeichenmethode der Klasse **`TreeCanvas`**, die *Punktdarstellung* und die *Punkt-* und *Bereichseingabe* werden durch Methoden der Klasse **`InputArea`** ausgeführt.

Für die *PUNKTMENGE* müssen wir weitere Klassen aufnehmen.

- Wir wählen einen **`Point2DVector`** zum Speichern der Punkte in der Eingabereihenfolge als Spezialisierung der Java-eigenen Klasse **`Vector`**, die bereits Methoden zum Hinzufügen und Löschen von Objekten enthält.
- Für den 2-d-Baum benötigen wir zusätzlich sortierbare Punktlisten. Dafür sehen wir die Klasse **`Pointlist`** vor. Sie stützt sich auf der Java-eigenen Klasse **`LinkedList`** ab.

Bei einer Orientierung an der gewählten Programmarchitektur, dem MVC-Konzept, sind die Klassen **`Tree`**, **`Point2DVector`** und **`Pointlist`** als Modell-Klassen einzuordnen. **`InputArea`** und **`TreeCanvas`** gehören zu den View-Klassen für die grafische Benutzeroberfläche. Die Mausaktionen, die noch der Klasse **`InputArea`** zugeordnet sind, werden für das Zusammenspiel nach dem MVC-Konzept in separate Controller-Klassen ausgelagert. Einzelheiten zur Implementierung der Benutzeroberfläche und damit auch zur noch fehlenden Klasse **`Menü`** folgen im Kapitel 8.2.1.

Zunächst ergeben unsere Überlegungen die vorläufigen Klassendiagramme von Abbildung 43 und Abbildung 44.

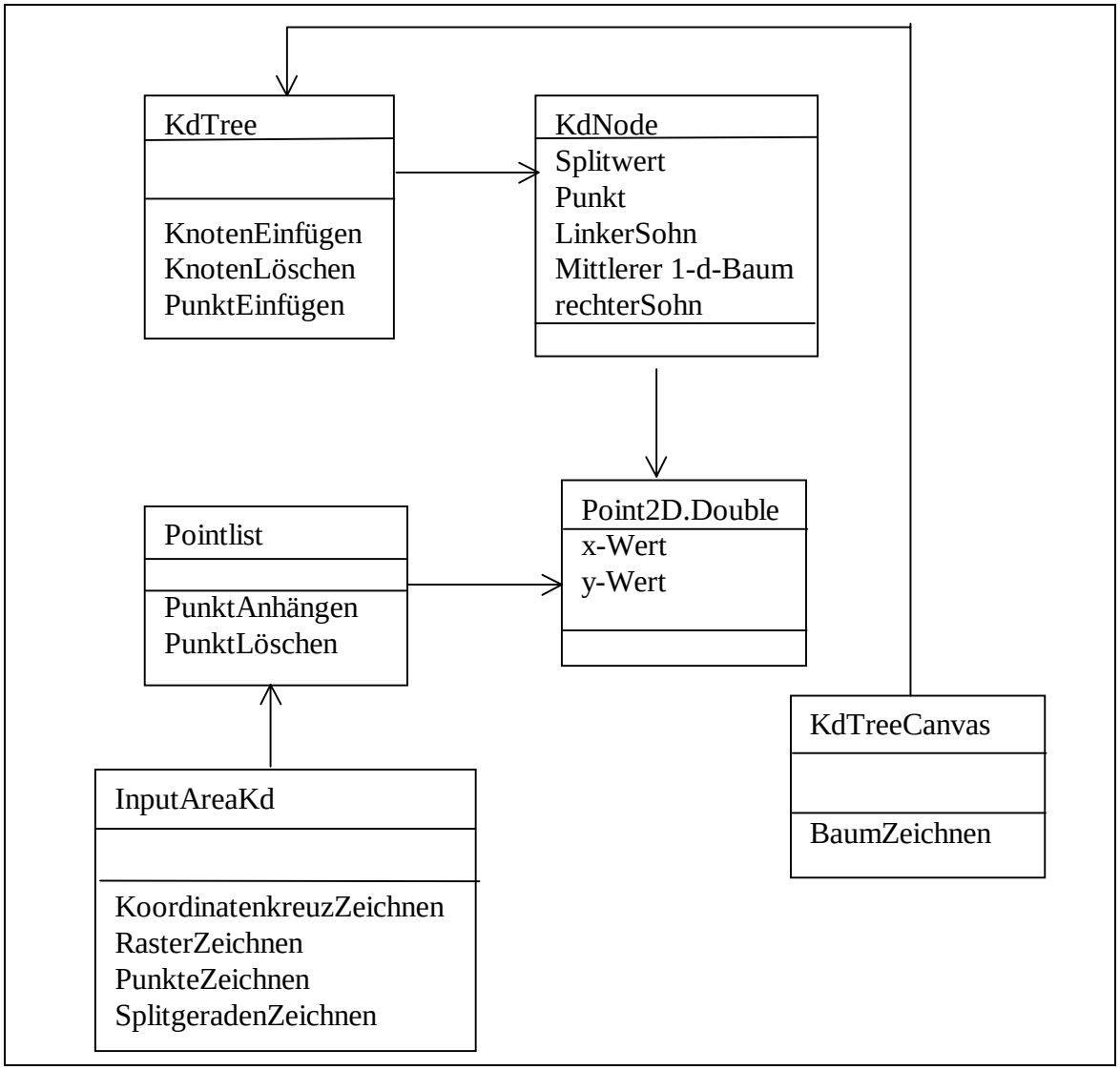


Abbildung 43: Vorläufiges Klassendiagramm zum 2-d-Baum[66]

[66] Zur Erläuterung der Symbole siehe Tabelle A10

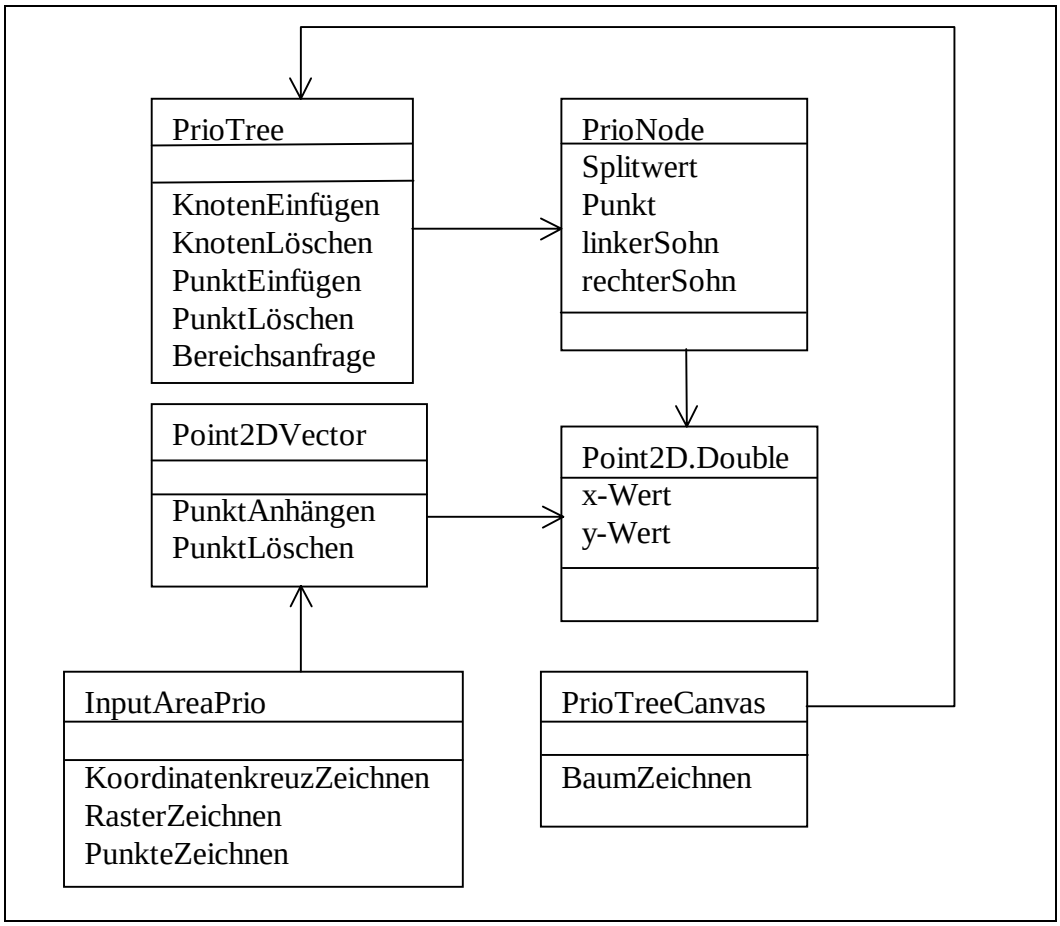


Abbildung 44: Vorläufiges Klassendiagramm zum Prioritätssuchbaum[67]

Ein Blick auf die beiden Klassendiagramme zeigt offensichtliche Übereinstimmungen. Im Kapitel 8 werden wir deshalb als Verfeinerung gemeinsam nutzbare Klassen und eine Baumhierarchie festlegen.

[67] Zur Erläuterung der Symbole siehe Tabelle A10

8. Construction / Erzeugung

Aufbauend auf den Ergebnissen der Analysephase geht es hier um die Verfeinerung der Entwürfe und deren Implementation in Java. Dabei wird die Vorgehensweise in dieser Phase exemplarisch beschrieben und nur auf besondere Einzelprobleme eingegangen.

8.1 Iterative und inkrementelle Vorgehensweise

Die in dieser Phase durchzuführenden Arbeiten erfolgen iterativ. Dies bedeutet, dass wir uns jeweils zunächst einen geeigneten Anwendungsfall heraussuchen und diesen einschließlich der erforderlichen Tests komplett fertig stellen. Danach erarbeiten wir nach und nach die anderen Anwendungsfälle, bis das gesamte Programm zur Verfügung steht. Der Umfang des fertigen Programms nimmt durch die Iterationen stetig zu. Es wächst inkrementell. Dies gilt nicht unbedingt in gleichem Maße für den Codeumfang, da beim Ausführen weiterer Anwendungsfälle bereits fertige Klassen wiederverwendet werden können und Gemeinsamkeiten erkannt werden. Diese führen nachträglich zur Bildung von Superklassen, so dass der Programmcode bei dieser iterativen Vorgehensweise ständig überarbeitet wird.
Zunächst beschreiben wir das Erstellen der gemeinsamen Benutzerschnittstelle, unsere grundsätzliche Entscheidung, wie die Bäume gezeichnet werden sollen und die Verfeinerung der Baumklassenbildung. Im Anschluss folgen Erklärungen zu den gemeinsam genutzten Methoden bezüglich Drucken, Laden, Speichern und Hilfesystem. Abschließend gehen wir auf die speziellen Entwicklungen zum 2-d-Baum und zum Prioritätssuchbaum ein.

8.2 Entwicklung der gemeinsamen Komponenten

Nach unserer Entscheidung für zwei getrennte Programmteile zum Arbeiten mit dem 2-d-Baum und dem Prioritätssuchbaum, geht es zunächst darum, eine gemeinsame einheitliche Arbeitsumgebung zu schaffen.

Diese soll auch als Testumgebung bei der weiteren Programmentwicklung genutzt werden.
Somit sind die Komponenten für ein Basisfenster zu realisieren, die in beiden Spezialfenstern eingesetzt werden können. Dort müssen dann nur die spezifischen Anteile ergänzt werden.

8.2.1 Entwicklung der Benutzeroberfläche

Mit Hilfe der in Java 2 enthaltenen Pakete **`javax.swing.*`** und **`javax.swing.event.*`** gestalten wir die gemeinsame Basis für die Benutzeroberfläche zu den Suchbäumen.
Die Swing-Pakete sind eine Erweiterung des AWT aus Java 1.1. Diese neuen Klassen sind im Gegensatz zu den AWT-Komponenten nicht mehr auf betriebssystemabhängige Benutzerschnittstellenelemente angewiesen. Sie zeichnen selbst alle Schnittstellenelemente in Java und verwalten diese.
Die Swing-Pakete bieten daher eine Vielzahl von komfortablen Schnittstellenelementen zum Erzeugen von Benutzeroberflächen, die nun frei gestaltet werden können. Einschränkungen durch das jeweilige Betriebssystem brauchen dabei nicht bedacht zu werden.
Für Menüsysteme stellt Swing die Klassen **`JMenu`** und **`JMenuItem`** zur Verfügung. Mit ihnen lassen sich Menüsysteme in Applikationen entwickeln, die auch völlig unproblematisch in Applets benutzt werden können.
Das Paket **`javax.swing.event.*`** stellt die für die Realisierung des MVC-Konzeptes benötigten Listener-Klassen für die Eventsteuerung zur Verfügung.

Unsere Basis für die Abbildung 45 ist ein Swing-Fenster der Klasse **`JFrame`**, das unter der Titelleiste das übliche Zeilenmenü mit den in der Analyse beschriebenen Menüpunkten aufweist. Darunter schließt sich eine Buttonleiste mit Schaltflächen für schnell ausführbare Aktionen an, z. B. zum Anhalten animierter Darstellungen.
Unter der Buttonleiste bleibt der Hauptteil des Fensters für den Eingabebereich und den Ausgabebereich reserviert.
Am unteren Fensterrand wird eine Statuszeile eingefügt. In dieser können Benutzerhinweise und Fehlermeldungen ausgegeben werden.

Der Eingabe- und Ausgabebereich sollen individuell veränderbar sein und dies nicht nur abhängig von der Gesamtfenstergröße. So soll der Ausgabebereich das Betrachten von Bäumen unterschiedlichster Größe ermöglichen. Dies wird über einen scrollbaren Ausgabebereich ermöglicht, der es dem Benutzer überlässt, welchen Baumausschnitt er gerade betrachten möchte.

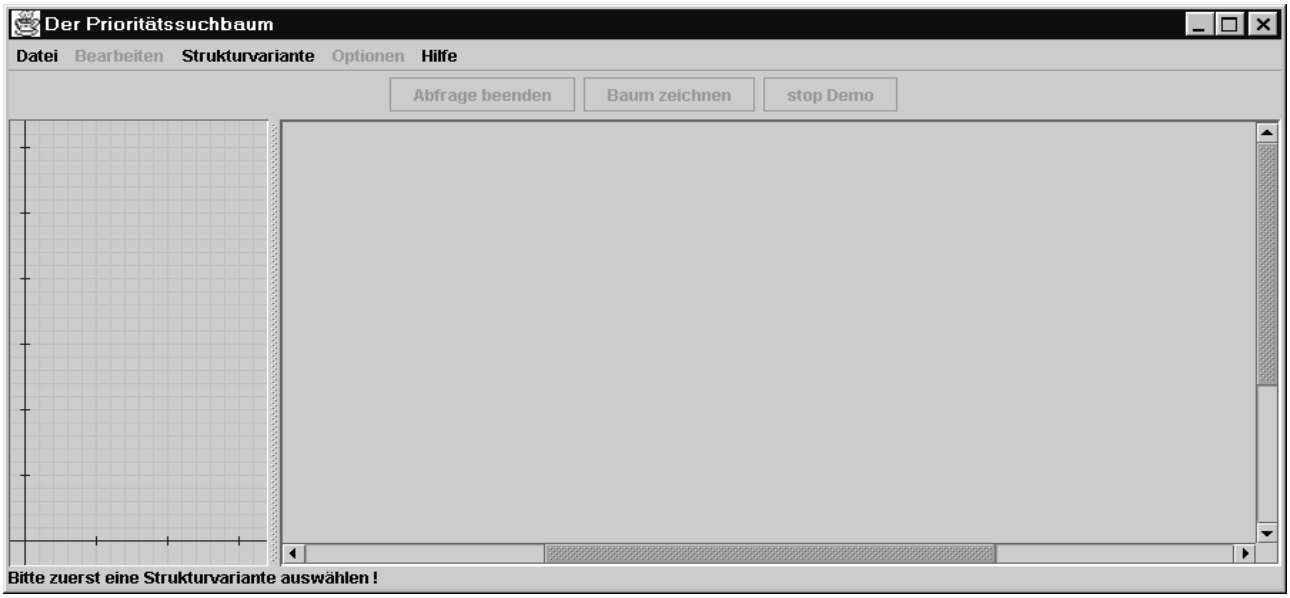


Abbildung 45: Basisfenster für die Suchbäume

Der mittlere Fensterabschnitt mit dem Eingabe- und dem Ausgabe-Bereich wird als eigene Klasse entworfen. Sie stellt einen zweigeteilten Bereich zur Verfügung, in den individuell **`JPanel`**-Klassen zum Zeichnen und für Benutzeraktionen eingebunden werden können. Die beiden Bereiche sind durch einen Mittelsteg voneinander getrennt. Dieser kann mit der Maus nach links und rechts verschoben werden. So lassen sich individuelle Veränderungen an der Aufteilung zwischen dem Eingabe- und dem Ausgabebereich vornehmen.

Die Menüs werden für den 2-d-Baum und den Prioritätssuchbaum als getrennte Klassen entwickelt, da sie unterschiedliche Untermenüpunkte enthalten. Das beiden Menüs gemeinsame Dateimenü wird zuvor als eigene Klasse erstellt und dann in die speziellen Menüs eingebunden.

Für die Buttonleiste wird eine gemeinsame Klasse eingesetzt. Sie stellt die benötigten Schaltflächen zur Verfügung.

In der Eingabefläche weisen die Fenster für den 2-d-Baum und den Prioritätssuchbaum stets das in Abbildung 45 sichtbare Erscheinungsbild für den 1. Quadranten mit Koordinatenkreuz und Rasterung auf.

Während der Arbeit in diesen Fenstern müssen im Eingabebereich aber unterschiedliche zusätzliche Darstellungen vorgenommen werden. Dies betrifft die Ausgabe der Splitgeraden beim 2-d-Baum und die Darstellung von Anfragerechteck und Anfragehalbstreifen.

Es empfielt sich die Entwicklung einer gemeinsamen Oberklasse **`InputArea`** mit den Spezialisierungen **`InputAreaKd`** und **`InputAreaPrio`**. Der scrollbare Ausgabebereich weist weiter keine Gemeinsamkeiten auf. Die Klassen **`KdTreeCanvas`** und **`PrioTreeCanvas`** werden spezifisch entwickelt.[68] Einen Überblick aller an der Oberfläche beteiligten Klassen zeigt Abbildung 46.

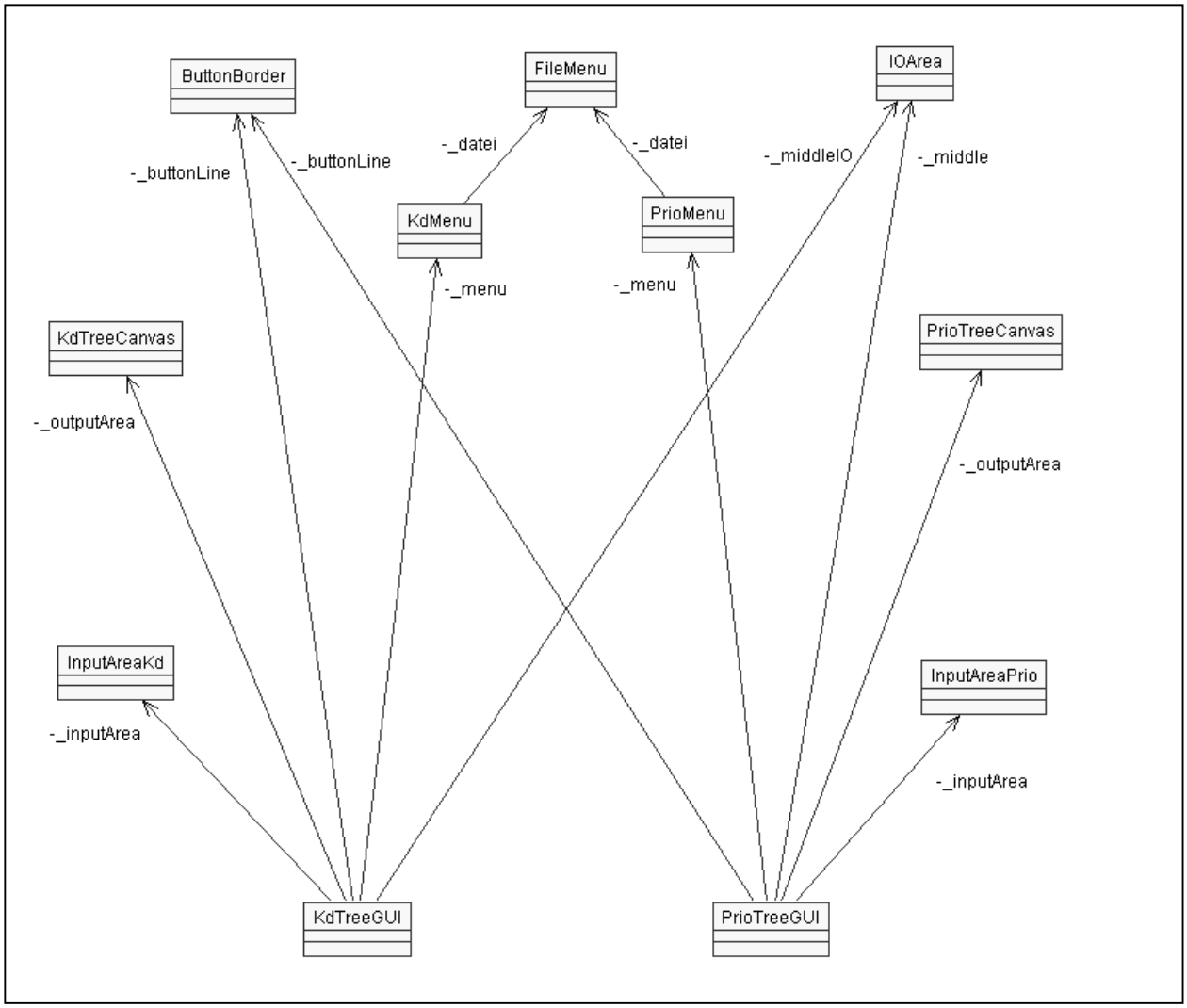


Abbildung 46: Klassendiagramm zur Benutzeroberfläche[69]

[68] Siehe hierzu auch Kapitel 8.3 und Kapitel 8.4

[69] Zur Erläuterung der Symbole siehe Tabelle A10
Zum Erstellen des Diagramms wurde die Demoversion des Programms Rational Rose 98 eingesetzt.

Beim Gestalten der konkreten GUI-Fenster für den 2-d-Baum und den Prioritätssuchbaum müssen dann nur noch die beschriebenen Komponenten in der gewünschten Form zusammengestellt werden. Abbildung 47 zeigt dies stellvertretend für den Prioritätssuchbaum.

```
        // Fentertitel weitergeben an Superklasse JFrame
        super(windowTitle);
        // auf Fensterkontext zugreifen
        Container c = getContentPane();
        // Layout-Manager BorderLayout verwenden
        c.setLayout(new BorderLayout());
        // Menue einbinden
        _menu = new PrioMenu(this);
        // Buttonleiste einbinden
        _buttonLine = new ButtonBorder(this);
        c.add(_buttonLine, BorderLayout.NORTH);
        _buttonLine.setVisible(true);
        _buttonLine.disableAll();
        // Trennfeld mit Ein-Ausgabebereich mittig einfuegen
        _middle = new IOArea(_inputArea, _outputArea);
        c.add(_middle, BorderLayout.CENTER);
        // Statuszeile am unteren Rand hinzufügen
        _statusLine.setForeground(Color.black);
        _statusLine.setText("Bitte zuerst eine Strukturvariante"
                            + " ausw\u00E4hlen !");
        c.add(_statusLine, BorderLayout.SOUTH);
```

Abbildung 47: Zusammenstellen der Fensterkomponenten

Dabei sind *_inputArea* und *_outputArea* Instanzen der spezifischen Klassen **InputAreaPrio** und **PrioTreeCanvas**. Beide Klassen sind Subklassen der Swingklasse **JComponent** und können deshalb problemlos an den Konstruktor der Klasse **IOArea** übergeben werden:

```
public IOArea(JComponent left, JComponent right)
```

Die gemeinsam nutzbaren Klassen haben wir in das Paket **common** aufgenommen. Es wird in die speziellen Pakete **kd** und **prio** importiert.

Zur Realisierung des MVC-Konzeptes müssen den ereignisliefernden Komponenten der Benutzeroberfläche nun noch Controller zugeordnet werden, um die Benutzeraktionen zu registrieren und die geforderten Aktionen im jeweiligen Modell auszulösen. Abbildung 48 zeigt das Zusammenspiel der Klassen am Beispiel des 2-d-Baums.

Der Eingabefläche wird ein MouseListener zur Registrierung von Mausklicks und ein MouseMotionListener zur Kontrolle von Mausbewegungen hinzugefügt. Dies geschieht über separate Controller-Klassen. Sie werten die ausgelösten Maus-Events aus und rufen die benötigten Methoden in den Modell-Klassen auf.
Der Controller für die Menüaktionen wird in die zentrale Klasse GUI aufgenommen, in der das Zusammenstellen aller Komponenten erfolgt.
Die Modelle Baum und Punktliste ihrerseits werden von weiteren Controllern überwacht, die nach jeder Modelländerung dafür sorgen, dass die Ansichten (View) neu gezeichnet werden.
Da alle Controller als Vermittler zwischen Oberfläche und Modell auftreten, verwenden wir die Bezeichnung Adapter in ihren Klassennamen.

Folgende Adapter–Klassen werden realisiert :

- Adapterklasse zwischen Eingabefläche und Modellen zur Umsetzung der MouseEvents:
  `InputAreaToTreeAndPointsAdapter`
- Adapterklasse zwischen Eingabefläche und Statuszeile zur Kommentierung von MouseEvents:
  `InputAreaToStatuslineAdapter`
- Adapterklassen zwischen Modellen und Ausgabefläche
  Zwischen Punktmenge und Eingabebereich:
  `PointVectorToViewAdapter` bzw **`PointlistToViewAdapter`**
  Zwischen Suchbaum und Baumzeichenfläche:
  `KdTreeToViewAdapter` bzw **`PrioTreeToViewAdapter`**

Der Controller der Eingabefläche **`InputAreaToTreeAndPointsAdapter`** analysiert die Mausaktionen des Benutzers. Der im Eingabebereich ausgelöste MouseEvent wird überprüft.
Aus dem Event lässt sich der angeklickte Punkt ermitteln und überprüfen. Ist der Punkt unter den eingestellten Programmoptionen erlaubt, werden die Einfügemethoden in der Baumklasse und in der Punktmengenklasse aufgerufen. Die Datenmodelle Punktmenge und Prioritätssuchbaum werden aktualisiert.
Die Veränderungen in den Modellen sorgen ihrerseits dafür, dass über die bei den Modellen angemeldeten ChangeListener **`PointListToViewAdapter`** und Kd**`TreeToViewAdapter`** das Neuzeichnen der Punktmenge und des Suchbaumes veranlasst wird.

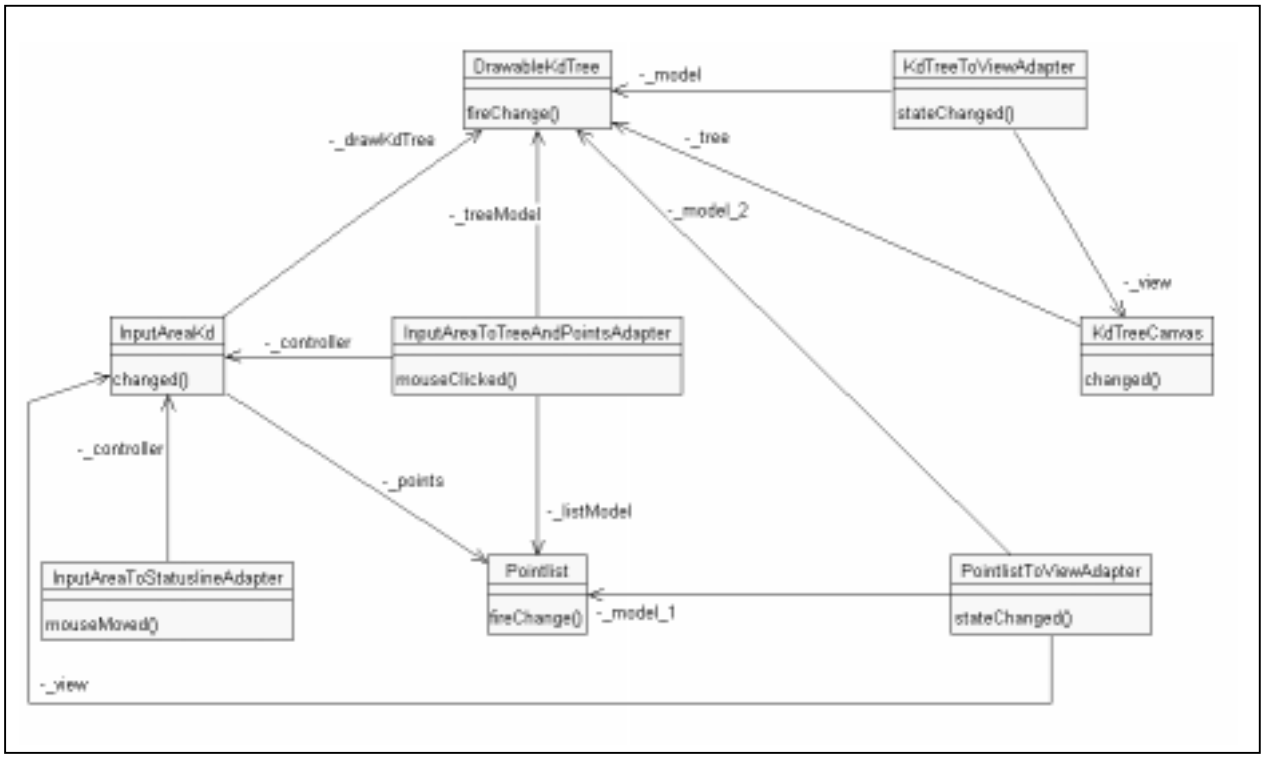


Abbildung 48: Zusammenspiel der Klassen nach dem MVC-Prinzip[70]

Alle Methoden in den Modellklassen von Abbildung 48, die zu einer Veränderung des aktuellen Modells führen, rufen die Methode **`fireChange()`** auf. Sie benachrichtigen die angemeldeten ChangeListener-Klassen **`PointlistToViewAdapter`** und **`KdTreeToViewAdapter`** über den Aufruf der Methode **`stateChanged()`** von der Änderung. Diese Methode übergibt den aktuellen Zustand des Modells zum Neuanzeigen an die View-Klassen **`InputAreaKd`** und **`KdTreeCanvas`**. Dort erfolgt in der Methode **`changed()`** die Neuanzeige durch **`repaint()`**.

Ein Beispiel, wie ein ChangeListener bei einer Modell-Klasse angemeldet wird, zeigt das Interaktionsdiagramm in Abbildung 49. Der Adapter zwischen Punktvektor und Eingabebereich wird zunächst instantiiert. Er wird als ChangeListener bei der Punktmenge angemeldet, dort in die Liste der ChangeListener eingetragen.

[70] Zur Erläuterung der Symbole siehe Tabelle A10
Zum Erstellen des Diagramms wurde die Demoversion des Programms Rational Rose 98 eingesetzt.

Die Punktmenge als Modell sorgt danach sofort dafür, dass der neue Controller die zugehörige View-Klasse über die Änderung informiert. Diese zeichnet die aktuelle Punktmenge erneut in den Eingabebereich ein.

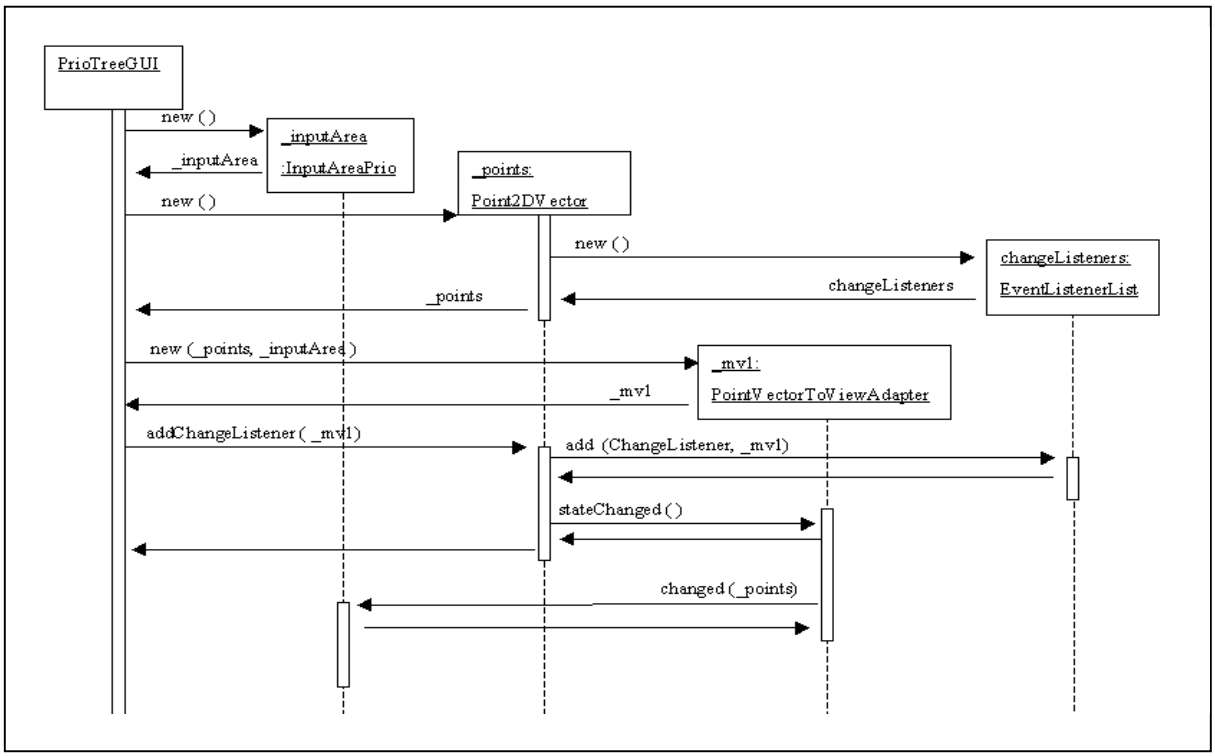


Abbildung 49: Interaktionsdiagramm zum Anmelden eines ChangeListeners[71]

8.2.2 Algorithmus zum „schönen" Zeichnen der Suchbäume

Ein weiteres gemeinsames Problem ist die Entscheidung, wie die Suchbäume in der Baumzeichenfläche dargestellt werden. Die Bäume sollen stets gleichmäßig aussehen, auch wenn dynamisch Veränderungen am Baummodell vorgenommen werden. Dies bedeutet, dass die Bäume in der Zeichenfläche aktualisiert werden müssen.

Wir haben uns für die Verwendung des Algorithmus von Reingold und Tilford aus deren Artikel "Tidier Drawings of Trees" [REIN81] entschieden.

[71] Zur Erläuterung der Symbole siehe Tabelle A11

Die Grundlage für das Zeichnen ist eine stets aktuelle Zuordnung von x- und y-Koordinaten zu jedem Knoten des Baumes. Damit kann der Baum in einem Preorder-Baumdurchlauf jederzeit „schön“ in die Ausgabefläche gezeichnet werden.

Was genau heißt „schön” zeichnen?

Reingold und Tilford greifen in ihrem Artikel zunächst auf Anforderungen an das „schöne” Zeichnen zurück, die bereits von Wetherell & Shannon [WETH79] aufgestellt wurden:

- Die Knoten einer Ebene liegen alle auf einer Höhe.
- Die Ebenen befinden sich parallel untereinander.
- Der linke Sohn ist links vom Vater positioniert.
- Der rechte Sohn befindet sich rechts vom Vater.
- Der Vater liegt eine Ebene oberhalb der Söhne mittig zwischen ihnen.
- Der Baum spiegelt also die Durchlaufanordnung der bekannten Traversal-Algorithmen Inorder, Preorder oder Postorder wider.
- Der Baum hat minimale Breite.

Nach dem Algorithmus von Wetherell & Shannon werden Teilbäume, abhängig von ihrer Position im Baum, jedoch manchmal unterschiedlich dargestellt, weil die Symmetrie von Bäumen noch nicht berücksichtigt wird.
Diese Schwäche behebt der hier verwendete Algorithmus von Reingold und Tilford.
Teilbäume werden danach immer gleichförmig gezeichnet, unabhängig von ihrer Position im Baum und ihrer Umgebung.
Der Preis für diese natürlichere Darstellung ist jedoch manchmal eine etwas größere Baumbreite.

Um die Anforderungen zu erfüllen, bestimmt der R&T-Algorithmus relative Positionen der Knoten in den Teilbäumen. Erst zum Zeichnen werden diese relativen Angaben in konkrete x- und y-Koordinaten umgerechnet, wenn die gewünschte Position des Wurzelknotens im Zeichenbereich bekannt ist.
Das Ermitteln der relativen Positionen geschieht während eines Postorder-Durchlaufs.

Die beiden Teilbäume eines Knotens werden zunächst unabhängig voneinander untersucht, die am weitesten außen liegenden Knoten dabei in einer separaten Datenstruktur festgehalten (Extrempositionen). Anschließend werden beide Teilbäume unter Berücksichtigung eines Minimalabstandes zwischen den Knoten so dicht wie möglich nebeneinander angeordnet. Dabei werden die neuen Knotenpositionen (Offsets) festgehalten und zwar relativ zu ihrem Vater, der über den Wurzeln der Teilbäume mittig positioniert wird.

Väter stehen mittig oberhalb der Söhne, deshalb reicht ein Distanzwert in jedem Knoten, der den horizontalen Offset zwischen dem Vaterknoten und jedem seiner Söhne speichert.

Die Abbildung 50 zeigt die roten Offsetwerte der Knoten vor und nach dem Hinzufügen eines neuen x-Wertes in einen Blattsuchbaum. Der Mindestabstand zwischen zwei Knoten einer Ebene beträgt hierbei 60. Beim Berechnen der tatsächlichen x-Koordinatenwerte zum Zeichnen führt dies zu den blauen Werten, wenn für die Wurzel der Wert 700 übergeben wird. Blattknoten der tiefsten Ebene haben den Offsetwert 0.

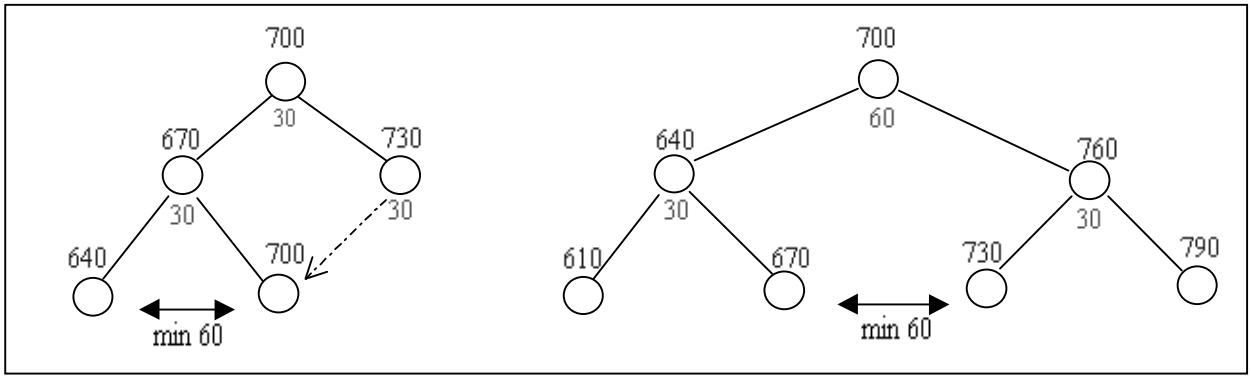


Abbildung 50: Schönes Zeichnen mit relativem Offset und berechnetem x-Wert

Offset und x-Koordinate könnten mit einer Speicheradresse auskommen, der Klarheit zuliebe verwendet der Algorithmus aber separate Variablen für Offset und x-Koordinate.

Jede Knotendatenstruktur muss also für das Zeichnen der Bäume um x- und y-Wert und Offset erweitert werden. Für die Attribute zum Zeichnen erstellen wir die separate Klasse **`PaintInfo`**, die zur jeweiligen Knotenklasse hinzugefügt wird, wenn zeichenbare Knoten benötigt werden.

Das Ermitteln der relativen Offset-Werte erfolgt nach jeder Veränderung am zeichenbaren Baum durch die Methode **`setup()`**, die von der verändernden Methode aufgerufen wird. Das Umrechnen der Offsets in konkrete x- und y-Koordinaten zum Zeichnen erfolgt durch die Methode **`petrify()`**, an die die x-Koordinate des Wurzelknotens als Parameter übergeben wird (siehe Abbildung 50).

8.2.3 Baumklassen

Die offensichtlichen Gemeinsamkeiten zwischen dem 2-d-Baum und dem Prioritätssuchbaum haben uns dazu bewogen, die nachfolgend beschriebene Struktur aufzustellen.
Da beide Bäume Blattsuchbäume sind, schaffen wird die Superklasse **`LeafSearchTree`**, die als gemeinsame Methoden z. B. den Selektor für den Wurzelknoten, die An- und Abmeldemethoden für den ChangeListener und **`fireChange()`** enthält.
Genauso verfahren wir für die Baumknotenklassen. **`SearchTreeNode`** wird Superklasse für **`KdNode`** und **`PrioNode`** (Abbildung 51) und enthält die üblichen Knotenattribute für den Splitwert und den linken und rechten Sohn.

Weiter müssen wir zwischen diesen „normalen“ Modellklassen und den zeichenbaren Klassen unterscheiden. Nur die zeichenbaren Klassen haben für den Einsatz in unserem Programm Bedeutung. Die ihnen zugrunde liegenden normalen Baumklassen sind zusätzlich für die Nutzung der Datenstrukturen in sonstigen Anwendungen gedacht.
Damit ein Knoten zeichenbar wird, muss er um die Zeicheninformation[72] x-Koordinate, y-Koordinate und Offset ergänzt werden. Dazu wird die Klasse **`PaintInfo`** eingebunden, und wir erhalten die „drawable“-Knotenklassen. Die Baumklassen werden zu zeichenbaren Klassen, indem die Methoden **`setup()`** zum Ermitteln des Offsets und **`petrify()`** zum Berechnen der Zeichenkoordinaten ergänzt werden. Zusätzlich müssen alle Methoden überschrieben werden, die mit zusätzlicher Zeicheninformation versehen werden. Z. B. wird die Methode für die Bereichsanfrage geändert. Um die durchlaufenen Wege festzuhalten, wird dabei in jedem besuchten Knoten *`_marked`*, das Attribut der Zeicheninformation, auf *`true`* gesetzt.
So entstehen die „drawable“-Baumklassen. Auch den **`LeafSearchTree`** brauchen wir als zeichenbare Klasse, um ihn im **`KdTree`** bei der allgemeinen Strukturvariante als mittleren Baum einsetzen zu können.
Abbildung 51 zeigt die implementierte Klassenstruktur für die zeichenbaren Bäume. Alle gemeinsam nutzbaren Klassen haben wir dem Paket **`common`** hinzugefügt. Es enthält jetzt insgesamt die Klassen von Tabelle 2.

[72] Siehe Kapitel 8.2.2 Schöner Zeichnen

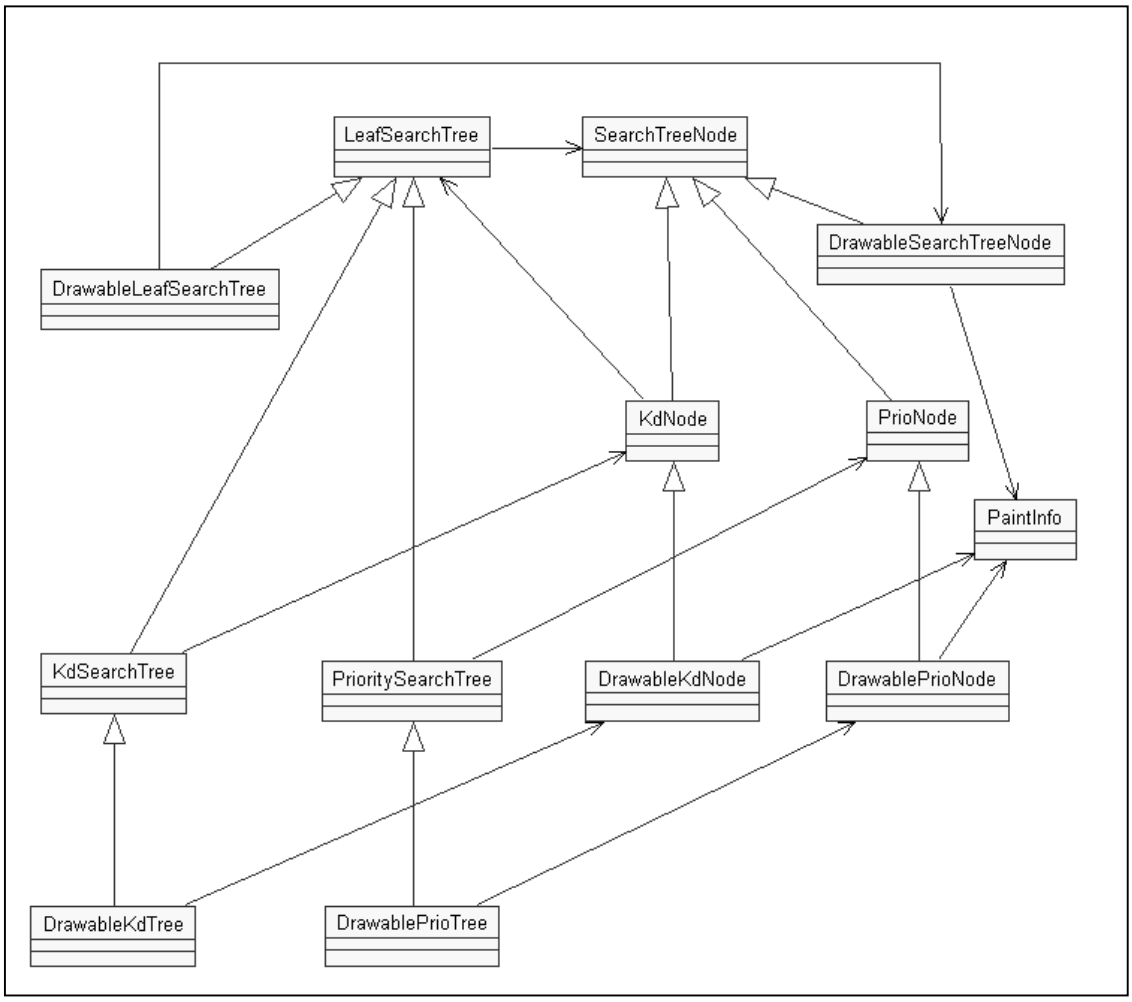


Abbildung 51: Klassendiagramm der implementierten Baumklassen[73]

DrawableLeafSearchTree	ButtonBorder	StartWindow
DrawableSearchTreeNode	FileMenu	StartWindowApplet
LeafSearchTree	IOArea	SuffixFilterJPG
SearchTreeNode	Point2DVector	TreeException
PaintInfo	Pointlist	VisualGeomApplet

Tabelle 2: Klassen aus dem Paket `common`

[73] Zur Erläuterung der Symbole siehe Tabelle A10
Zum Erstellen des Diagramms wurde die Demoversion des Programms Rational Rose 98 eingesetzt.

8.2.4 Drucken

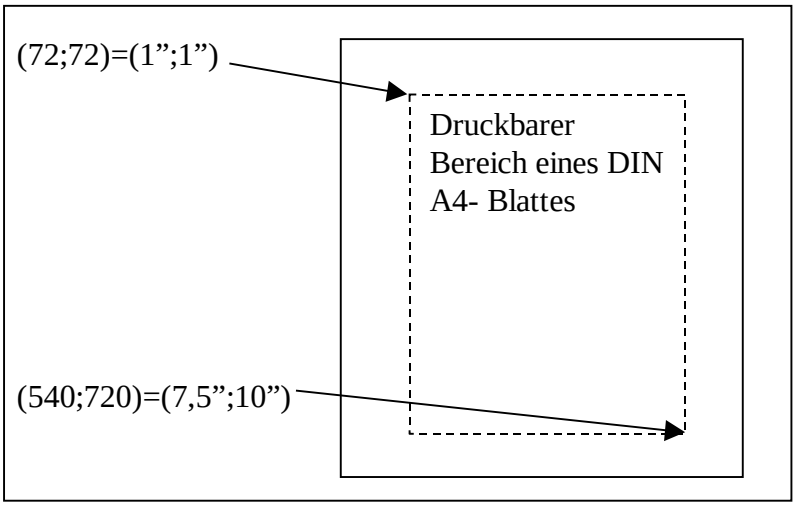


Abbildung 52: Druckbereich eines DIN A4-Blattes

Das Implementieren einer Druckroutine gestaltete sich entgegen eigenen Erwartungen schwierig. Hierzu trug u. a. bei, dass sich der Druckertreiber unter den verschiedenen Betriebssystemen nicht identisch verhält. Schließlich mussten wir feststellen, dass unter Windows95 und 98 bei einem Pixelformat von 1/72” die sich aus dem Blattformat ergebenden 648 Bildpunkte in der Höhe nicht genutzt werden können. Experimente lieferten einen maximal zulässigen aber nicht nachvollziehbaren Wert von 514 Bildpunkten. Das Durchforsten der FAQs[74] und der News von SUN lieferte erst Ende Oktober 1999 den Hinweis, dass es sich hierbei um ein Bug[75] handelt. Er verhindert das Einstellen des Seitenformates unter Windows. Unter LINUX und Solaris trat dieses Problem nicht auf.[76]

In allen drei Fällen musste allerdings (nachvollziehbar in den Newsgruppen) festgestellt werden, dass der Zeitaufwand beim Drucken sehr hoch ist. Dabei wurden Daten im Umfang mehrerer Megabyte erzeugt. Dies konnte beim Drucken mehrerer Seiten in einer Postscriptdatei, die etwa 100MB groß war, festgestellt werden. Damit wird auch deutlich, warum die Druckmethode zwingend den Aufruf der Garbage-Collection erforderlich macht.

---

[74] Siehe hierzu A.1.3

[75] Dieser als Bug 4067405 spezifizierte Fehler soll mit dem Erscheinen der Version 1.3 beseitigt sein.

[76] Wir haben deshalb im Quellcode in den Dateien **`KdTreeGUI`** und **`PrioTreeGUI`** in der Druckmethode für den Ausgabebereich die korrekte Methode für die Ermittlung der Seitenlänge implementiert und danach mit einer weiteren Anweisung die Seitenlänge auf 514 Bildpunkte reduziert. Diese Anweisung braucht also nur auskommentiert zu werden.

Ein weiteres Problem beim Drucken ist, dass der Ausgabebereich abhängig von der Baumhöhe über mehrere Seiten ausgedruckt werden muss. Dies wird dadurch ermöglicht, dass das Image in mehrere Rechtecke, die jeweils auf eine DIN-A4-Seite passen unterteilt wird.

8.2.5 Speichern und Laden der Punktmenge

Da die zu speichernde Punktmenge nur für den Einsatz in diesem Programm von Nutzen ist, haben wir keinen besonderen Wert auf das Datenformat der Dateien gelegt. Die Koordinatenwerte werden zeilenweise abgespeichert und eingelesen. Die Endungen **kd** bzw. **kda** für die Punktlisten des 2-d-Baums und **pr** bzw. **pra** für die Punktlisten des Prioritätssuchbaums sind fest voreingestellt. Sie kennzeichnen Punktmengen mit „disjunkten“ Werten und allgemeine Punktmengen.

8.2.6 Auswahl des Grafikformates

Standardmäßig unterstützt JDK nicht das Speichern von Images in einem Grafikformat. Für das Speichern von Eingabebereich und den Ausgabebereichen boten sich deshalb zwei Lösungen an:

- JAI **J**ava **a**dvanced **i**maging
  Diese Paket ist erst im Juli 1999 von der Firma SUN für den allgemeinen Gebrauch freigegeben worden. Es ermöglicht das Laden und Speichern von Bildern in verschiedenen Grafikformaten, hat jedoch den Nachteil der Evaluation[77], wie SUN es selbst beschreibt. Damit verbietet sich die Nutzung dieses Paketes, da ja unser Programm von den Endbenutzern verwendet werden soll.

[77] Es befindet sich trotz Freigabe noch in der Entwicklung. Die Klassen- und Methodenbezeichnungen können sich bei Freigabe der nächsten Release ändern.

- das Paket **`com.sun.image.codec.jpeg`**

  Diese Klassen sind zwar Bestandteil des JDK 1.2. Es fehlt allerdings eine Dokumentation. Desweiteren liegen sie in der Datei **`src.jar`** nicht in compilierter Form vor. Mit diesen Klassen wird u. a. das Speichern eines Image in eine Bilddatei im JPEG-Format ermöglicht.

  Wir haben uns entschieden, diese Klassen aus dem Paket **`com.sun.image.codec.jpeg`** zu verwenden. Dies bedeutete, dass wir sie compilieren mussten und dann als **`jar`**-Datei **`codec.jar`** für die Nutzung auf allen drei Betriebssystemen gepackt haben.

  Der Ablauf beim Speichern des Eingabebereichs oder der Ausgabebereiche lässt sich in folgende Schritte untergliedern:

  1. Es wird aus dem Grafikbereich ein Buffered-Image erzeugt.
  2. Zum Buffered-Image wird ein Grafikkontext eingerichtet.
  3. In das Buffered-Image wird die Grafik gezeichnet.
  4. Ein BufferedOutputStream mit einem FileOutputStream(<Dateiname>) wird eingerichtet.
  5. Es wird eine Instanz des JPEGImage-Encoders erzeugt. Diese ist zum Encodieren der Image-Daten als JPEG-Datenfluß erforderlich.
  6. Der Vorgang des Encodierens wird gestartet.
  7. Der Puffer wird in die Datei geschrieben und dann geschlossen.

8.2.7 Das Java-Hilfesystem

Mit **`JavaHelp`** bietet die Firma Sun ein gelungen strukturiertes Hilfesystem, mit dem sogar kontextbezogene Hilfen implementiert werden können. Dabei werden XML-Dateien für die Beschreibung des Hilfesystems und HTML-Dateien für die eigentliche Hilfeinformation verwendet. Da dies mit einem selbst entworfenen Hilfesystem nicht möglich gewesen wäre, haben wir uns für den Einsatz des Hilfe-Toolkits der Firma Sun entschieden. Es bietet es folgende Vorteile:

- Volltextsuche

  Im Java-Hilfesystem ist eine eigene Suchdatenbank implementiert. Sie kann aus den Hilfedaten generiert werden. Bestandteil ist dabei auch ein Tool für die Indizierung der Such-Datenbank.

- Kontextsensitive Hilfe

  Es lassen sich Hilfeinformationen im unmittelbaren Zusammenhang mit der aktuellen Situation der Applikation realisieren.
- Datenkompression

  Bei der Verwendung von JDK2 lassen sich sämtliche Hilfedateien zu einer **`jar`**-Datei zusammenfassen und komprimieren. Dies erleichtert das Verwalten der Hilfedateien.
- Veränderbar- und Ergänzbarkeit

  Unabhängig von der Applikation oder dem Applet kann das Hilfesystem jederzeit erweitert oder verändert werden.

Da das Hilfesystem noch nicht auf allen Plattformen implementiert ist, wurde auf die Nutzung der SearchEngine verzichtet. Wir haben deshalb nur die Verwaltung des Inhaltsverzeichnisses und des Indexverzeichnisses implementiert. Nur so konnten wir uns auf die ausschließliche Nutzung der in der Datei **`jhall.jar`** enthaltenen Klassen von **`JavaHelp`** zurückziehen und auch die Verwendung unter Linux gewährleisten.
Beim Einsatz von **`JavaHelp`** ergaben sich einige Ungereimtheiten im Zusammenhang mit Applets und dem Programm **`appletviewer`**.

- Unter Linux wird die Metadatei mit den Strukturinformationen (**`*.hs`**) nicht gefunden.
- Unter Windows erfolgt die Ausgabe einer Fehlermeldung, dass **`javax.help`** nicht aufgefunden werden kann.
- Im Debug-Modus der Entwicklungsumgebung KAWA ergaben sich keine Probleme.

Unter Windows konnte das Hilfesystem beim Starten des Applets aus dem Netscape-Communicator oder Internet Explorer problemlos genutzt werden.
Vielleicht liegt es daran, dass es laut mitgelieferter Dokumentation durchaus zu Schwierigkeiten im Zusammenhang mit dem JDK2 kommen kann.
Für die Entwicklung der Hilfedateien stand uns außer einem HTML-Editor kein weiteres Werkzeug zur Verfügung. Wir sind deshalb den Weg gegangen, die mitgelieferten Beispiele zu analysieren und hieraus für unsere Anwendung angepasste Dateien zu erzeugen.

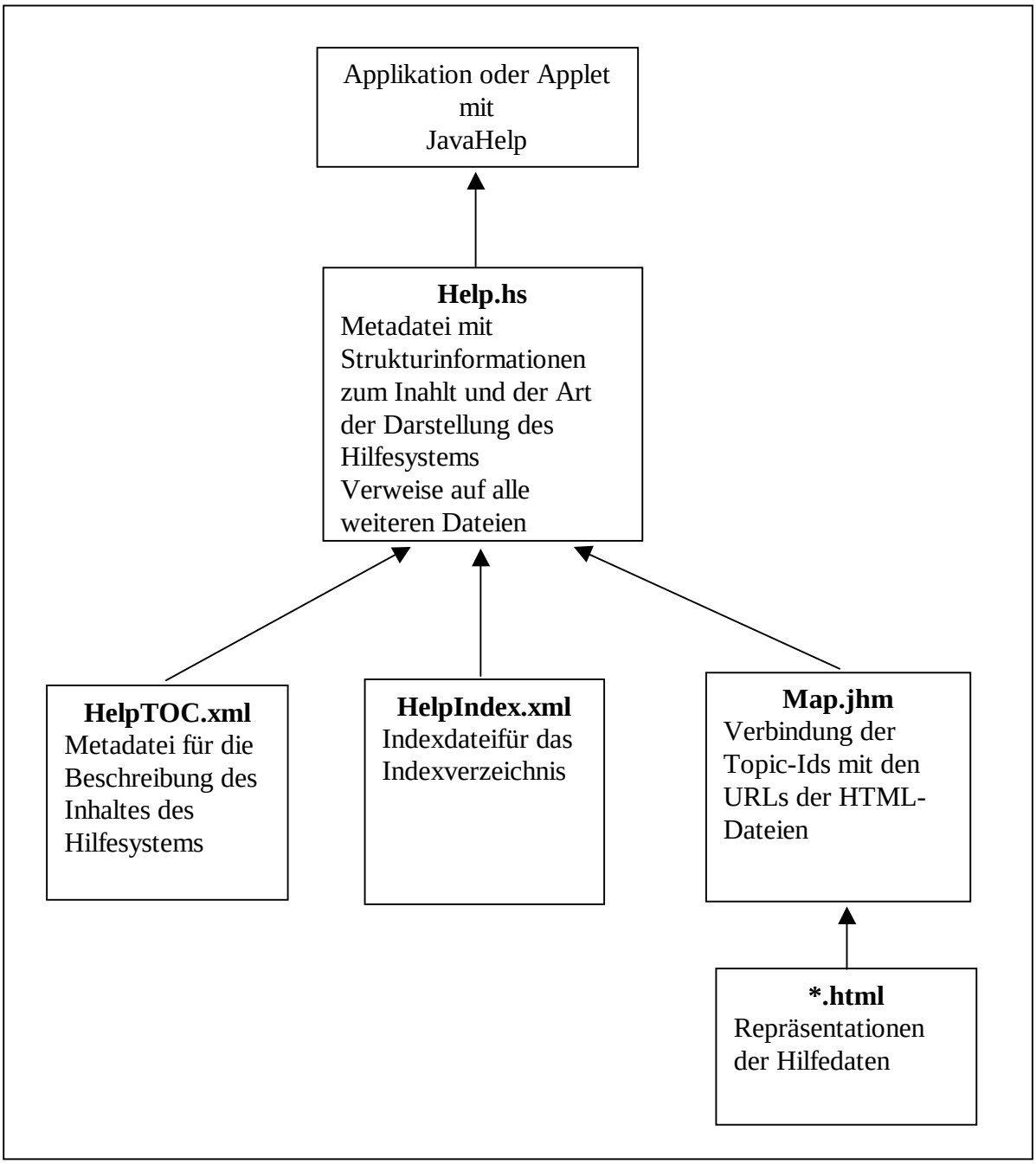


Abbildung 53: Grundsätzlicher Aufbau des Hilfesystems

Das Hilfesystem setzt sich also aus folgenden Dateien zusammen

- Meta-Dateien, mit denen die Struktur des Online-Hilfesystems festgelegt wird,
- Hilfedateien, die die Hilfeinformationen enthalten und
- das JavaHelp-API.

Der genaue Aufbau des Hilfesystems kann der

Abbildung 53 entnommen werden. Eine Kurzanleitung für die Implementation der Hilfedateien findet sich im Kapitel A.2 des Anhangs. Die Nutzung des Hilfesystems ist über den Menüpunkt „Hilfe-Hilfethemen" implementiert. Dabei wird nach der Anwahl des Menüpunktes in **`KdTreeGUI`** bzw. **`PrioTreeGUI`** die Methode **`_lookhelp()`** von Abbildung 54 ausgeführt.

```
/* Variablen fuer die Verwendung von JavaHelp                        */

/* Variable fuer das Objekt, das die die Kommunikation               */
/* zwischen der Applikation und dem HelpSet herstellen soll          */
private HelpBroker _helpbroker;
/* Variable fuer die URL                                             */
private URL        _helpURL;
/* Variable fuer das Objekt fuer das Herstellen einer Verbindung     */
/* zum Hilfesystem */
private HelpSet    _helpSet;
/* Angabe des Verzeichnisses und Namens der HelpSet-Datei relativ */
/* zum Programmverzeichnis                                           */
private String     _helpSetName = "/apps/geodast/kdHelp/kdHelp.hs";
/* Aufruf des Hilfesystems                                           */
private void _lookHelp()
{
   /* Kontrolle, ob eine Verbindung zum Hilfessystem besteht         */
   if (_helpSet == null)
   {
      try
      {
         /* Aufbau einer Zeichenkette, die aus dem absoluten         */
         /* Dateiverzeichnisses des Hilfesystems und dem             */
         /* Dateinamen der Hilfedatei besteht                        */
         String _helpURL = "file:"
                         + System.getProperty("user.dir")
                         + _helpSetName;
         /* Konvertierung der Zeichenkette in eine absolute URL      */
         URL url = new URL(_helpURL);
         /* Koppelung des Objektes mit den Daten des Hilfesystems */
         _helpSet = new HelpSet(null, url);
      }
      catch(Exception ex)
      {
         System.out.println(_helpSetName +" ist nicht korrekt"
                           + "oder nicht auffindbar!");
         return;
      }
   } // if
   /* Erzeugen eines Helpbrokers, der den Standardnavigator nutzt */
   _helpbroker = _helpSet.createHelpBroker();
   /* Darstellen des Helpbrokers                                    */
   _helpbroker.setDisplayed(true);
} // _lookHelp
```

Abbildung 54: Nutzen des HelpSet aus einer Anwendung

8.3 Construction des 2-d-Baums

Die Klasse **`DrawableKdTree`** enthält einen Konstruktor für alle drei Formen des 2-d-Baumes. Initialisiert wird dabei mit *`null`*.

Das iterative Umsetzen der Anwendungsfälle beim 2-d-Baum erfolgte in nachstehender Reihenfolge.

1. Einfügen von Punkten in einen dynamischen 2-d-Baum mit sofortiger Anzeige der Änderungen im Baum und der neuen Splitgeraden
2. Bereichssuche im dynamischen 2-d-Baum
3. Löschen von Punkten im dynamischen 2-d-Baum
4. Der sofortige Aufbau des ausgeglichenen 2-d-Baums
5. Der schrittweise Aufbau des ausgeglichenen 2-d-Baums
6. Der Aufbau des allgemeinen 2-d-Baums
   - Aufteilen allgemeiner Punktmengen, Darstellen der Splitgeraden und Kennzeichnen der Knoten mit einem 1-d-Baum
   - Einbinden separater Fenster für die 1-d-Bäume
7. Animationen:
   - für das Einfügen von Punkten
   - für die Bereichssuche
8. Demos

Jeder Iterationsschritt endete mit gezielten Tests. Exemplarische Testbeispiele sind im Kapitel 9.1 beschrieben.

8.3.1 Einfügen von Punkten im dynamischen 2-d-Baum

Beim Anklicken eines Punktes im Eingabebereich erfolgen zwei Aktionen.

Der Punkt wird an das Ende der Punktliste angefügt. Hierfür haben wir in der Klasse **`Point-list`** die Methode **`addPoint()`** implementiert. Mit dieser Methode wird der neue Punkt an das Ende der Liste angehängt.

Für das Einfügen eines Punktes in den dynamischen 2-d-Baum wurde der Einfügealgorithmus mit der Methode **`insertPoint()`** in der Klasse **`KdTree`** aufgenommen. Der Algorithmus bestimmt zunächst mittels **`_searchInsertPosition()`** rekursiv die Einfügeposition. Ist diese gefunden, wird mit der Methode **`_pointInsert()`** der Splitwert berechnet, ein neuer Splitknoten mit Splitwert und Splitrichtung erzeugt und ein neues Blatt mit dem Punkt in den 2-d-Baum eingefügt. Beide Knoten sind Instanzen der Klasse **`KdNode`**.

Nachdem das Einfügen erfolgreich getestet wurde, haben wir diese Methoden in den **`DrawableKdTree`** und den **`DrawableKdNode`** aufgenommen. **`insertPoint()`** wurde dabei um Methoden für die Zeicheninformationen und den Aufruf des ChangeListeners ergänzt. Da im Eingabebereich neben den Punkten zusätzlich auch die Splitgeraden gezeichnet werden sollen, haben wir in der Klasse **`DrawableKdNode`** zusätzlich das Attribut *`_line`* und Methoden für den Zugriff auf dieses Objekt aufgenommen, um in jedem Splitknoten Start- und Endpunkt der zugeordneten Splitgeraden zu speichern.

Mit der Eingabe eines Punktes erfolgt auch der Aufruf des ChangeListeners für die Punktmenge. Hierdurch wird der neue Punkt im Eingabebereich eingezeichnet. Danach erfolgt ein Preorderdurchlauf durch den dynamischen 2-d-Baum. Dabei wird auf die in den Blättern befindlichen Informationen für die Splitgeraden zugegriffen. Mit diesen Informationen werden die Splitgeraden im Eingabebereich eingezeichnet. Voraussetzung dafür ist allerdings, dass an die Eingabeklasse **`InputAreaKd`** neben der aktuellen Punktliste auch der 2-d-Baum übergeben wird.

In der Methode **`paint()`** der Baumausgabeklasse **`KdTreeCanvas`** wurde mit **`_treeDrawingKD()`**ein Preorderdurchlauf implementiert, um den dynamischen 2-d-Baum mit den abgespeicherten Punkten zu zeichnen. Bei sofortiger Darstellung werden alle Knoten und die Punkte schwarz gezeichnet.

Hinweis zur Implementation des dynamisch-interaktiven 2-d-Baums

Die **`insertPoint()-`**Methode wurde von uns überladen, indem wir neben den Punktkoordinaten auch noch zusätzlich den Splitwert übergeben haben.

Damit konnten die vorhandenen privaten Methoden für das Einfügen eines Blattes in den dynamischen 2-d-Baum durch geringe Ergänzungen, Fallunterscheidungen bei **`_pointInsert(),`** auch für den interaktiven dynamischen 2-d-Baum genutzt werden. Zusätzlich wurde in diesem Fall nur noch die Methode **`insertTest()`** ergänzt. Sie stützt sich auf den privaten Methoden für das Einfügen eines Punktes in den dynamischen 2-d-Baum ab und liefert die für das Testen der Benutzereingabe benötigte Splitinformation, also Splitrichtung und Splitintervall, zurück.

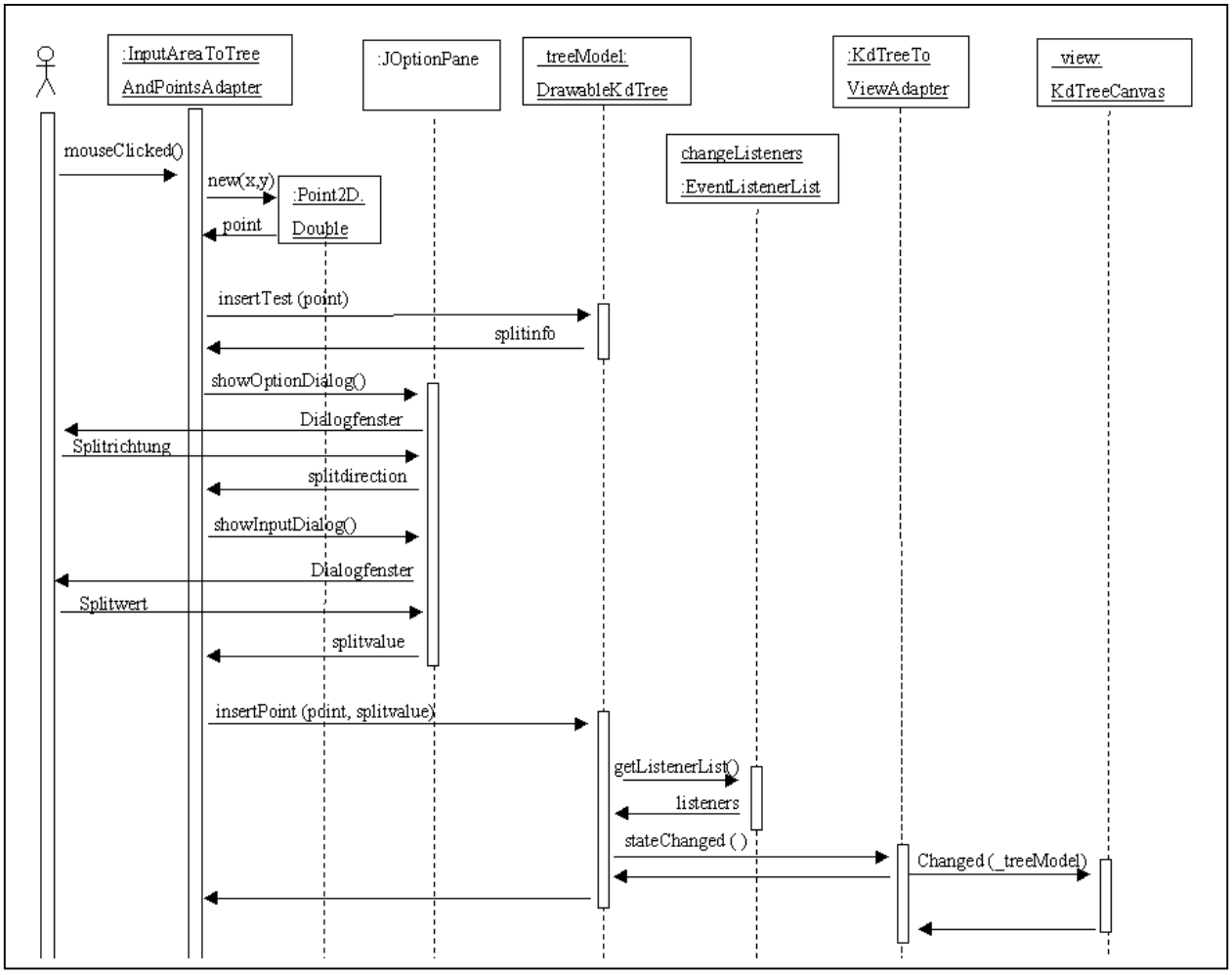


Abbildung 55: Interaktionsdiagramm zum dynamisch-interaktiven 2-d-Baum[78]

[78] Zur Erläuterung der Symbole siehe Tabelle A11

Hinweis zum Zeichenbereich

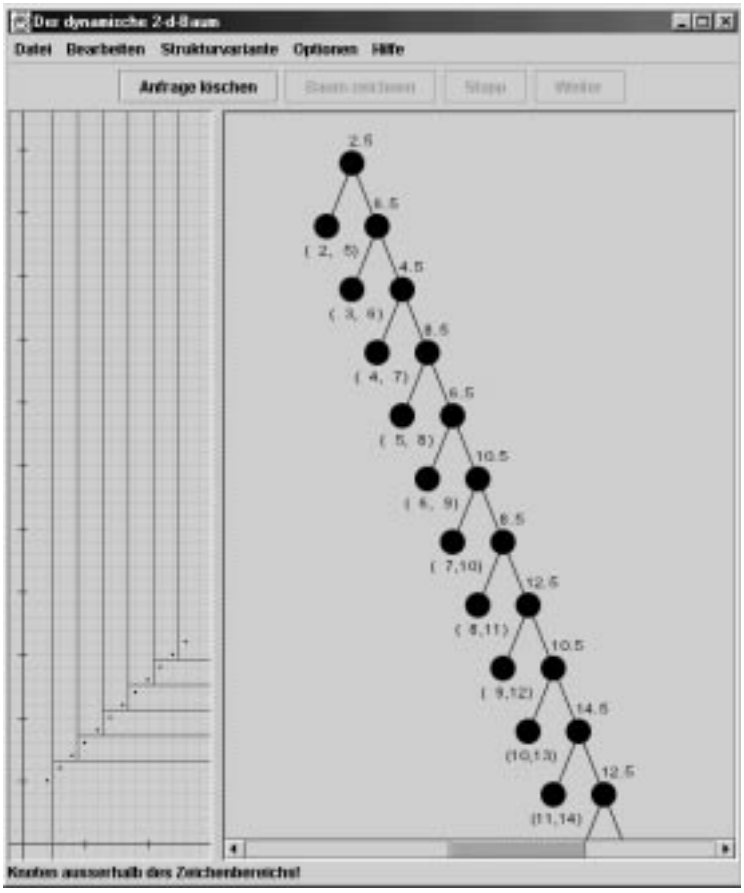


Abbildung 56: Knoten nicht im Zeichenbereich

Der Zeichenbereich für die Bäume ist beschränkt auf 1400 Bildpunkte in x- und 550 in y-Richtung. Dabei liegen die Knotenebenen im y-Abstand von 50 Bildpunkten. Der geringste Abstand in x-Richtung beträgt 40.

Gerade beim dynamischen 2-d-Baum kann dieser Zeichenbereich schnell verlassen werden. Deshalb haben wir die Methode **`controlDrawing()`** in der Klasse **`DrawableKdTree`** implementiert. Sie kontrolliert, ob der zu zeichnende Blattknoten noch innerhalb des darstellbaren Bereiches liegt und liefert den gefundenen Zustand als booleschen Wert zurück.

Der boolesche Wert ist *`false`*, wenn die Anzahl der Ebenen größer als 10 oder der x-Wert des zu zeichnenden Blattes außerhalb des Intervalls [0, 1400] liegt. Ist der zulässige Bereich überschritten, erfolgt eine Fehlerausgabe in der Statuszeile.

8.3.2 Bereichssuche im dynamischen 2-d-Baum

Um das Ergebnis der Bereichssuche darstellen zu können, wurde es erforderlich weitere Attribute und den Zugriff darauf in der Klasse **`PaintInfo`** zu ergänzen. Das Attribut *`_marked`* dient als boolesche Variable zur Information, ob ein innerer Knoten oder ein Blatt im Rahmen der Anfrage angefasst wurde. Mit dem Attribut *`_query`* wird kenntlich gemacht, ob das betreffende Blatt einen Punkt aus dem Anfragerechteck enthält.

In **`DrawableKdTree`** haben wir zwei weitere Methoden ergänzt. Die Methode **`findPoints()`** übernimmt das x- und das y-Intervall des Anfragerechtecks, setzt in allen besuchten Knoten *`_marked`* auf **`true`** und in allen Blättern, deren Punkte im Anfragerechteck liegen, zusätzlich die Variable *`_query`* auf **`true`**. Die sich im Anfragerechteck befindlichen Punkte werden in einer Punktliste eingetragen und diese an die aufrufende Klasse als Anfrageergebnis zurückgegeben.

Mit dem so modifizierten Baum können in der Klasse **`KdTreeCanvas`** die Markierungen für die Anfrage im Baum eingezeichnet werden. Die Ergebnisliste ermöglicht das Markieren der gefundenen Punkte im Eingabebereich **`InputAreaKd`**. Mit der Methode **`resetMarkedQuery()`** können die Attribute *`_marked`* und *`_query`* auf *`false`* zurückgesetzt werden.

8.3.3 Löschen von Punkten im dynamischen 2-d-Baum

Die grundsätzlichen Überlegungen zum Algorithmus finden sich bereits im Kapitel 7.4.3.2. Deshalb soll nachfolgend nur noch kurz auf die konkrete Implementierung eingegangen werden.

Für das Löschen haben wir in der Klasse **`DrawableKdTree`** die Methode **`deletePoint()`** implementiert. Da sie sowohl für den dynamischen als auch den dynamisch-interaktiven 2-d-Baum genutzt werden soll, haben wir eine Abfrage nach dem Baumtyp eingebaut. Damit kann diese Methode für beide Baumtypen genutzt werden.

Diese Methode ermittelt zunächst, ob sich überhaupt ein Blatt mit der betreffenden Punktinformation im Baum befindet. Ist dies der Fall, wird der 2-d-Baum von diesem Blatt ausgehend

nach oben durchsucht. Dabei werden alle Blätter und inneren Knoten gelöscht und die den Blättern zugeordneten Punkte in einer Punktliste gespeichert, bis der Splitknoten erreicht ist, an dem mit dieser Punktliste der neu aufzubauende Teilbaum anzuhängen ist.
Hierfür war es erforderlich, zusätzlich zu den Verweisen auf die Nachfolger auch einen Verweis auf den Vorgänger des Knotens zu implementieren. Dazu haben wir in der Klasse **`SearchTreeNode`** das Attribut **`_pre`** ergänzt und Methoden für den Zugriff auf dieses Attribut vorgesehen. Des weiteren wurde in der Klasse **`DrawableKdTree`** die private Methode **`_insertPoint()`** so erweitert, dass die Verweise auf die Vorgänger beim Einfügen eines neuen Blattes korrekt implementiert werden konnten.

8.3.4 Der ausgeglichene 2-d-Baum

Für den ausgeglichenen 2-d-Baum war die bereits implementierte Methode zum Einfügen eines Punktes nicht verwendbar. Wie der Algorithmus aus Kapitel 7.4.3.3 verdeutlicht, muss die Punktliste sukzessive sortiert, geteilt und erneut sortiert werden. Hierzu war es erforderlich, in der Klasse **`Pointlist`** zunächst die Methode **`addSortedXPoint()`**für das nach der x-Koordinate sortierte Einfügen eines Punktes in die Punktliste zu ergänzen. In der Klasse **`DrawableKdTree`** wurde die Methode **`insertPointList()`** aufgenommen, die aus der nach x-Koordinaten sortierten Punktliste den Baum aufbaut. Diese Methode ruft dazu eine private Methode auf, die den Splitwert berechnet, die Splitrichtung festlegt, die Koordinaten der zugeordneten Splitgeraden bestimmt, den Splitknoten erzeugt und die Punktliste in zwei Teillisten aufteilt.
Hatte die Ausgangsliste mehr als drei Elemente, so werden die Teillisten nach der zweiten Koordinate sortiert und die Methode erneut aufgerufen.
Bestand die Ausgangsliste nur noch aus zwei Punkten, werden Blattknoten gebildet und die Punkte in diese Knoten eingetragen.
Bestand die Ausgangsliste aus 3 Punkten, so befindet sich in einer der beiden Teillisten nur ein Punkt. Hierfür wird sofort der zugeordnete Blattknoten erzeugt, die anderen Teilliste nach der zweiten Koordinate sortiert und dann die Methode erneut aufgerufen.

8.3.5 Schrittweiser Aufbau des ausgeglichenen 2-d-Baums

Für den schrittweisen Aufbau des 2-d-Baums in der Ausgabefläche war es erforderlich, die angehängten Punktlisten im jeweiligen Knoten zwischenzuspeichern. Nur so konnten mit jedem Schritt diese Teillisten und die zugeordneten Splitgeraden ausgegeben werden (Abbildung 57). Zum Speichern der Listeninformation haben wir in der Klasse **`DrawableKdNode`** zusätzlich die Attribute *`_leftList`* und *`_rightList`* aus der Klasse **`Pointlist`** implementiert.

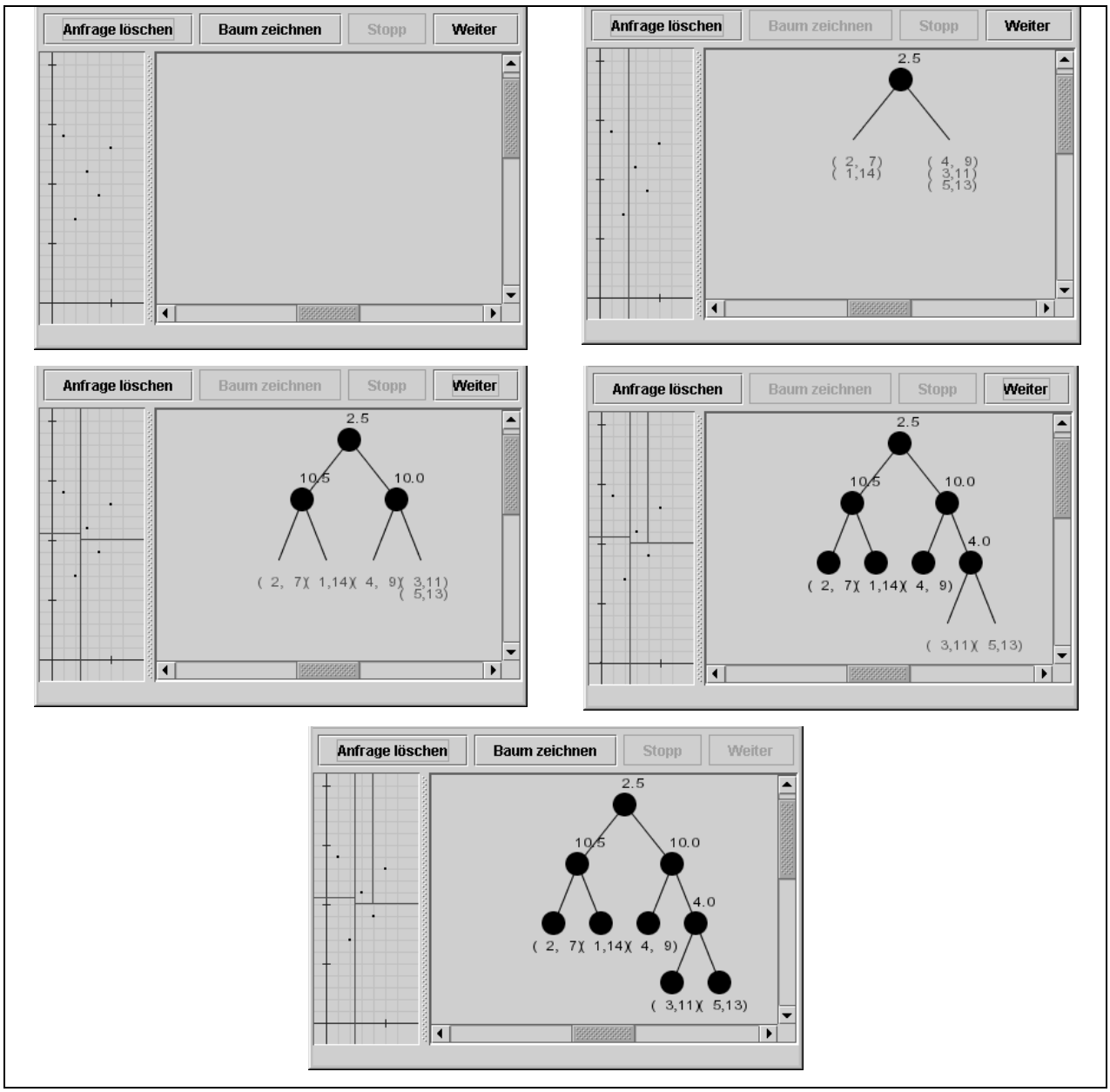


Abbildung 57: schrittweiser Aufbau des ausgeglichenen 2-d-Baums

Zwar ist damit die gesamte benötigte Information in den Splitknoten abgelegt, die Schwierigkeit stellte aber der Zugriff auf diese Daten dar, da der Baum ebenenweise aufgebaut werden muss. Hierzu wurde ein Breitendurchlauf und das Anlegen einer Hilfsdatenstruktur erforderlich. In der Klasse **`DrawableKdTree`** wurde dazu eine *`_queue`* ergänzt. Hierbei handelt es sich um ein Objekt aus der JDK-Klasse **`LinkedList`**. Am Ende der Methode **`insertPointlist()`** wird die private Methode für das Anlegen einer Queue mittels Breitendurchlauf ergänzt. In diese Queue werden alle Knoten Ebene für Ebene kopiert. Mit ihr wird der Aufbau nach Abbildung 57 ermöglicht.

Nach der Eingabe aller Punkte löst der Benutzer mit der Schaltfläche „Baum zeichnen“ den schrittweisen Aufbau aus. Dies bewirkt das Löschen von Baum und Splitgeraden und das Setzen eines Ebenenzählers auf Null. In den **`paint()`**-Methoden werden der Wurzelknoten mit den angehängten Teillisten und die Splitgerade dargestellt. Die angehängten Teillisten werden dabei in rot dargestellt. Mit dem Button „Weiter“ ist der Ebenenzähler inkrementierbar und damit die nächste Ebene darstellbar.
Ist die Anzahl der in der Queue befindlichen Knoten kleiner oder gleich $2^{Ebenenzähler+1}-1$, so wird der gesamte ausgeglichene 2-d-Baum dargestellt.

8.3.6 Der allgemeine 2-d-Baum

Bei der Implementation des 2-d-Baums für allgemeine Punktmengen traten zwei Probleme auf. Zum einen konnte der Algorithmus für den ausgeglichenen 2-d-Baum nicht genutzt werden, andererseits mussten aufgrund der Entscheidungen unter 7.4.3.5 Methoden für die Auswahl, das Öffnen und das Zeichnen der 1-d-Bäume implementiert werden.
Für das Sortieren der Punkte wurde es erforderlich, die Methoden **`addSortedXPoint()`** und **`addSortedYPoint()`** aus der Klasse **`Pointlist`** so zu erweitern, dass bei identischer x-Koordinate bzw. y-Koordinate die Punkte aufsteigend nach der y-Koordinate bzw. nach der x-Koordinate sortiert werden.

Aufteilen allgemeiner Punktmengen

Unabhängig vom implementierten Algorithmus ist der allgemeine 2-d-Baum in Sonderfällen nicht mehr ausgeglichen.

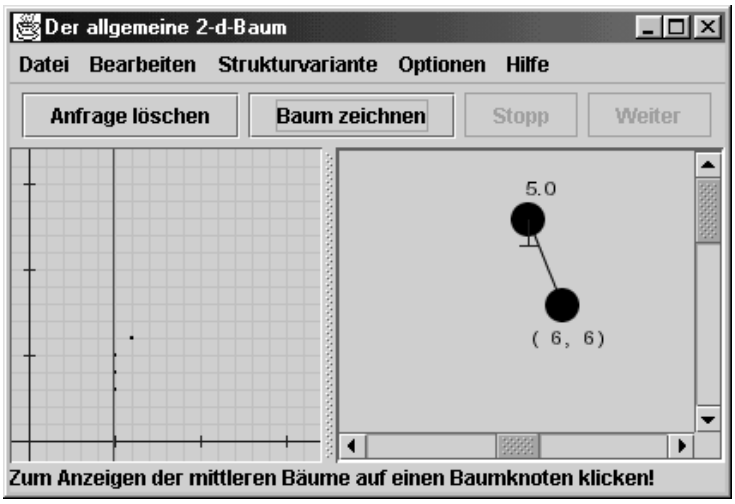
Der von uns implementierte Algorithmus für das Aufteilen der Punktliste in zwei Teillisten ermittelt zunächst das arithmetische Mittel der x-Koordinaten der in der „Mitte“ liegenden benachbarten Punkte. Da aber weitere Punkte mit dem berechneten Splitwert als x-Koordinate existieren können, wird im nächsten Schritt die Punktliste nach derartigen Punkten durchsucht. Diese Punkte müssen aus der vorhandenen Punktliste entfernt und in eine separate Punktliste eingetragen werden, aus der ein 1-d-Baum aufgebaut wird. Nur die verbleibende Restpunktliste darf unter Berücksichtigung des Splitwertes in zwei Teilpunktlisten aufgeteilt werden. Hierbei waren diverse Sonderfälle zu berücksichtigen.

Nachfolgend werden diese Sonderfälle kurz dargestellt. Alle werden in der privaten Methode **`_insertPointlistEqual()`** von **`DrawableKdTree`** durch Fallunterscheidungen berücksichtigt.

Alle Punkte liegen auf der Splitgeraden.

Nach dem Entfernen aller Punkte auf der Splitgeraden ergibt sich eine leere verbleibende Punktmenge.

Abbildung 58: leere Restpunktliste



Nach dem Entfernen der Punkte auf der Splitgeraden bleibt nur noch ein Punkt übrig.

Damit ist eine Teilpunktliste leer, die andere enthält nur einen Punkt.

Dieser Punkt muss rechts oder links am Splitknoten angehängt werden

Abbildung 59: Restpunktliste mit einem Punkt

Nach dem Entfernen aller Punkte auf der Splitgeraden enthält die Restpunktliste nur noch zwei Punkte. Hierbei sind zwei weitere Fälle zu unterscheiden:

- Der eine Punkt liegt rechts und der andere links von der Splitgeraden (Abbildung 60).
- Beide Punkte liegen rechts oder links von der Splitgeraden (Abbildung 60).

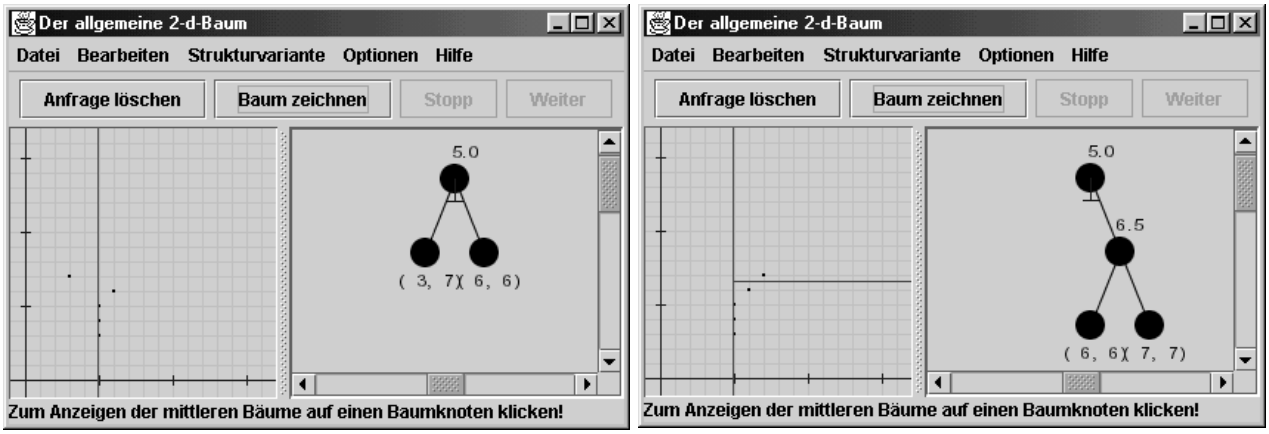


Abbildung 60: Restpunktliste mit zwei Punkten

Nach dem Entfernen aller Punkte auf der Splitgeraden enthält die Restpunktliste nur noch drei Punkte. Hierbei sind zwei weitere Fälle zu unterscheiden:

- Alle drei Punkte liegen rechts oder links von der Splitgeraden (Abbildung 61).
- Zwei Punkte liegen auf der einen und ein Punkt auf der anderen Seite der Splitgeraden (Abbildung 61).

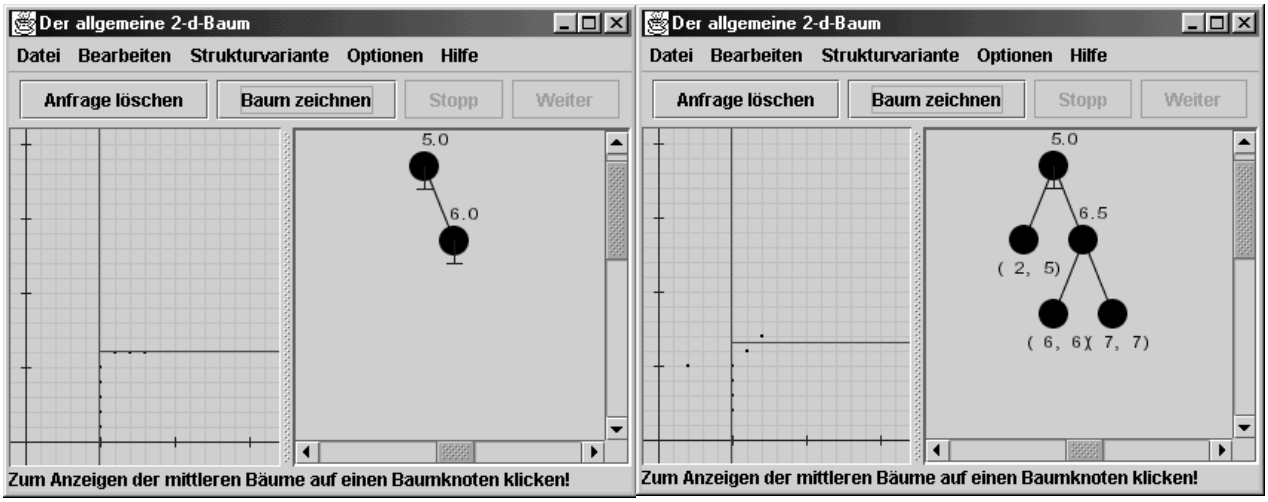


Abbildung 61: Restpunktliste mit drei Punkten

Nach dem Entfernen aller Punkte auf der Splitgeraden enthält die Restpunktliste mehr als drei Punkte. Hierbei sind weitere Fälle zu unterscheiden:

- Alle Punkte liegen auf einer Seite der Splitgeraden. Dieser Fall entspricht prinzipiell dem rechten Bild von Abbildung 61.
- Auf der einen Seite der Splitgeraden liegt nur ein Punkt auf der anderen die übrigen Punkte (Abbildung 62).
- Auf jeder Seite der Splitgeraden befinden sich mindestens zwei Punkte (Abbildung 62).

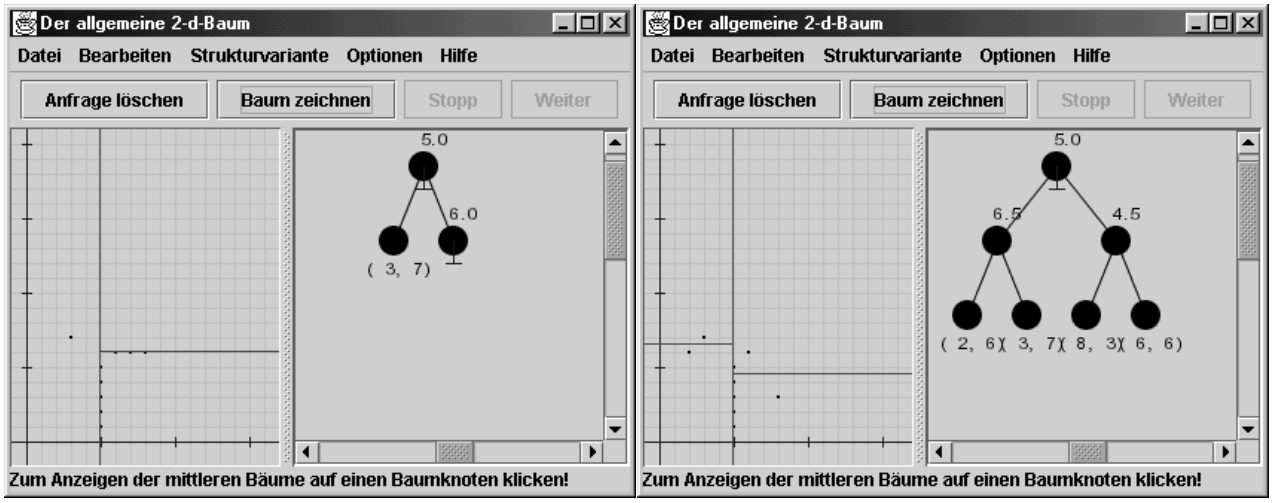


Abbildung 62: Restpunktliste mit mehr als drei Punkten

Kennzeichnen der Knoten, an denen 1-d-Bäume hängen

Die **`paint()`**-Methode in **`KdTreeCanvas`** berücksichtigt, ob sich am gerade zu zeichnenden Splitknoten ein 1-d-Baum befindet. Ist dies der Fall, wird durch die Methode **`_drawReferenceLeafSearchTree()`** eine blaue Markierung als Hinweis auf den 1-d-Baum eingezeichnet.

Einbinden separater Fenster für die 1-d-Bäume

Für die Reaktion auf Mausaktionen in der Baumzeichenfläche musste ein zusätzlicher Controller beim **`KdTreeCanvas`** angemeldet werden. Dadurch kann der Benutzer jetzt einen Knoten im 2-d-Baum anklicken, um dessen mittleren Blattsuchbaum zu betrachten. Dieser Controller ermittelt den x- und y-Wert des angeklickten Punktes in der Zeichenfläche. Damit durchsucht er den **`DrawableKdTree`**, um den angeklickten Knoten zu ermitteln.

Abbildung 63 zeigt, dass in Sonderfällen ein unausgeglichenen 2-d-Baum entsteht.

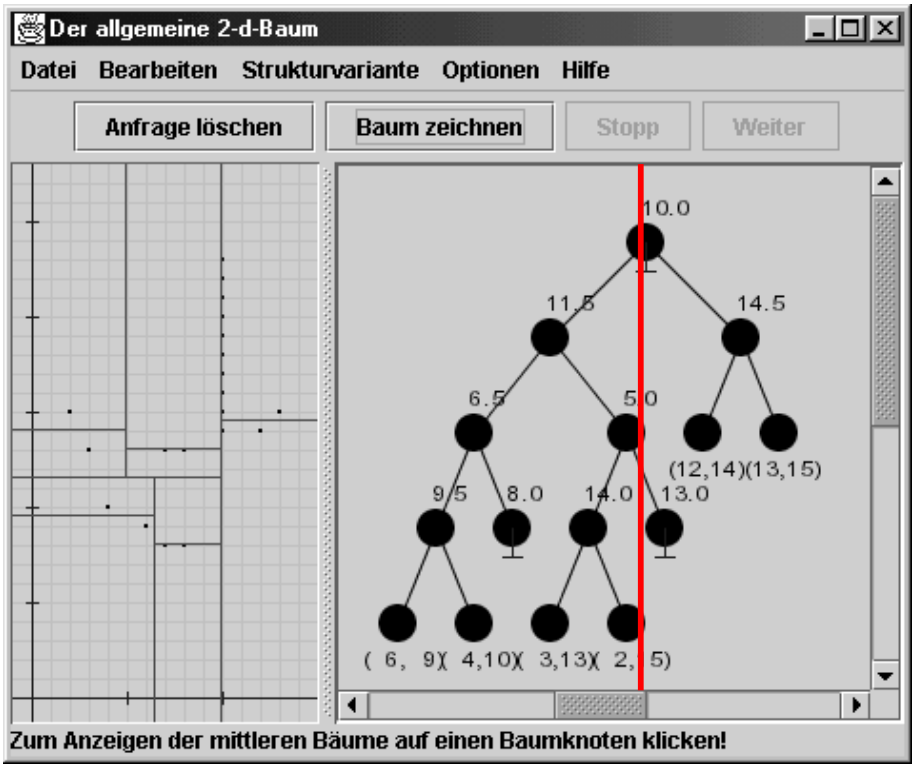


Abbildung 63: unausgeglichene Form des 2-d-Baums

Weil es im unausgewogenen Baum möglich ist, dass ein Knoten im linken Teilbaum einen größeren x-Wert (Zeichenkoordinate) aufweist als der Wurzelknoten, reicht es nicht aus, sich am x-Wert zu orientieren, um den richtigen Knoten zu finden. Zusätzlich muss die Tiefe des Baumes (y-Wert) berücksichtigt werden.

Wird z. B. der Knoten mit dem Splitwert 13 in Abbildung 63 gesucht, so ist dessen x-Wert größer als der x-Wert des Wurzelknotens. Eine normale Suche nach diesem Knoten würde im Wurzelknoten nach rechts verzweigen. Der rechte Teilbaum der Wurzel weist aber nicht die erforderliche Tiefe auf, deshalb wählt der Algorithmus zunächst den linken Nachfolger (y=11,5) und erst dann dessen rechten Sohn (x=5,0), weil jetzt auch der rechter Teilbaum die erforderliche Tiefe aufweist.

Dieser Suchalgorithmus wurde als Methode **`searchTree()`** in der Klasse **`DrawableKdTree`** ergänzt. Er liefert die Instanz eines 1-d-Baums zurück.
Ist im gefundenen Knoten ein nicht leerer 1-d-Baum vorhanden, so wird ein zusätzliches Fenster geöffnet und der Baum darin angezeigt. Dazu wurden eine zusätzliche Fensterklasse **`Middle-TreeFrame`** und eine weitere Klasse **`MiddleCanvas`** für die neue Zeichenfläche implementiert (Abbildung 64).

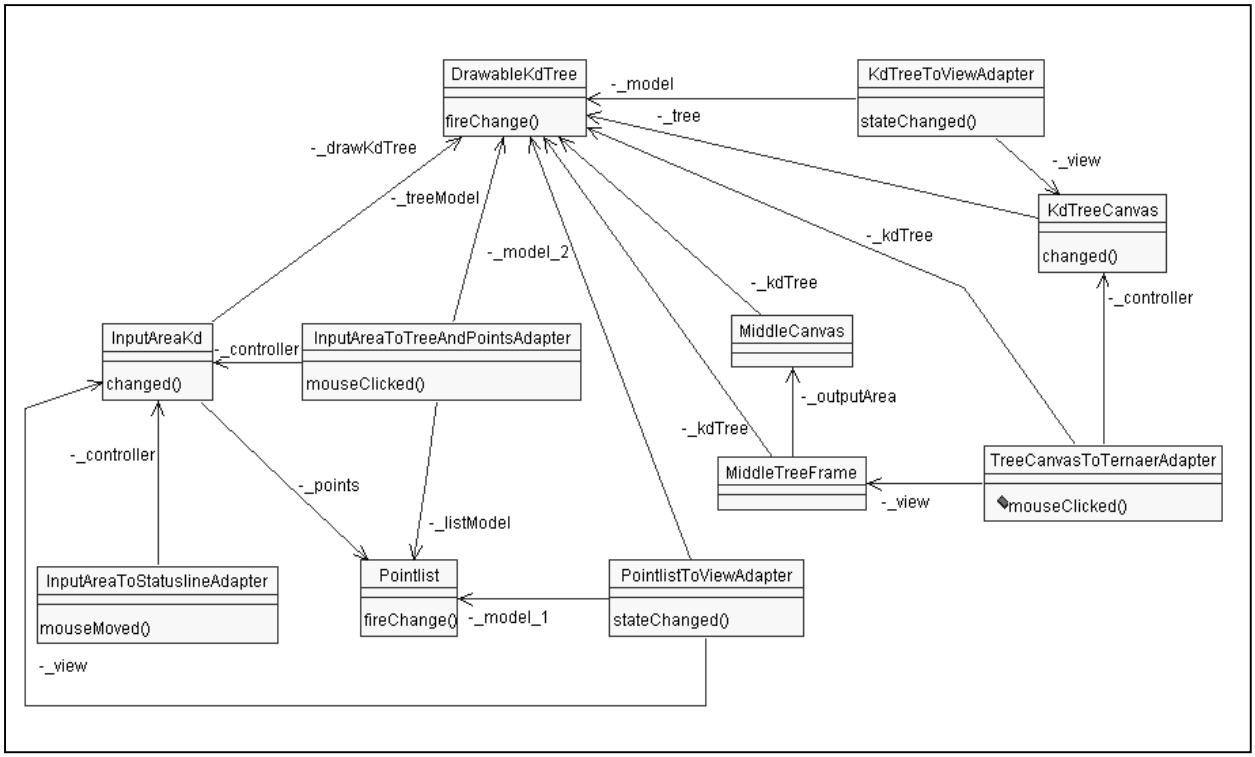


Abbildung 64: Klassendiagramm zum ternären 2-d-Baum[79]

[79] Zur Erläuterung der Symbole siehe Tabelle A10
Zum Erstellen des Diagramms wurde die Demoversion des Programms Rational Rose 98 eingesetzt.

8.3.7 Animierter Aufbau des 2-d-Baums

Im Kapitel 7.4. haben wir als Designentscheidung festgelegt, dass wir beim Einfügen von Punkten in den 2-d-Baum auf eine Animation per Bewegung verzichten. Die Visualiserung soll ausschließlich über den Farbeinsatz erfolgen. Insofern ist der Name Animation beim 2-d-Baum teilweise unpassend. Wegen einer einheitlichen Menüoberfläche haben wir uns allerdings dafür entschieden, wie beim Prioritätssuchbaum den Menüpunkt „animiert“ zu verwenden.

„Animiertes“ Einfügen im dynamischen 2-d-Baum

Die vom Algorithmus eingeschlagenen Wege im Baum werden rot eingezeichnet. Die beiden Punkte, die für die aktuelle Berechnung des neuen Splitwertes verwendet werden, erscheinen ebenfalls in rot. Dazu muss die Methode **`insertPoint()`** aus der Klasse **`DrawableKdTree`** so erweitert werden, dass in allen beim Einfügen angefassten Knoten das Flag *`_marked`* auf *`true`* gesetzt wird. Die **`paint()`**-Methode aus **`KdTreeCanvas`** fragt beim Zeichnen eines Knotens nun zusätzlich den Zustand der Variablen *`_marked`* ab.

Animiertes Einfügen im ausgeglichenen und allgemeinen 2-d-Baum

Für den animierten Aufbau eines ausgeglichenen oder allgemeinen 2-d-Baums wird auf die bereits implementierte schrittweise Darstellung zurückgegriffen. Nur wird jetzt der ebenenweise Aufbau des 2-d-Baums automatisch weitergeschaltet. Dies erfolgt durch einen Zähler-Thread, der nach einer festen Zeitspanne mit jedem neuen Schritt zusätzlich zu den bereits dargestellten Ebenen auch noch die nächste Ebene zeichnen lässt.
Der Thread kann stets definiert durch Betätigen des Buttons „Stopp“ beendet werden.

8.3.8 Demos zum 2-d-Baum

Aufgrund der bis hierher durchgeführten inkrementellen Softwareentwicklung stehen alle für die Demos zum dynamischen und zum ausgeglichenen 2-d-Baum benötigten Methoden zur Verfügung. Sie müssen nur in separaten Threads in der gewünschten Abfolge zusammengestellt werden. Dies erfolgt in den Thread-Klassen **`KdDemoDynamic`** und **`KdDemoStatic`**.

8.4 Construction des Prioritätssuchbaums

Der Prioritätssuchbaum wird in seiner dynamischen und semidynamische Ausprägung betrachtet. Deshalb enthält die Klasse **`DrawablePrioTree`** Konstruktoren für beide Formen des Baumes. Im dynamischen Fall wird der Baum mit einem speziellen Wurzelknoten initialisiert, der einen Wert ∞ enthält. Dieser ist größer als alle möglichen Eingaben für die Splitwerte [OTTM93]. Ohne diesen Spezialknoten ließe sich beim Löschen nicht immer ein Knotenpaar im Blattsuchbaum finden, sondern der größte Knotenwert wäre nur einzeln vorhanden. Dadurch entstünde ein Sonderfall, der immer abgeprüft werden muss. Mit dem zusätzlichen Startknoten ist diese Problematik ausgeräumt.
Im semidynamischen Fall wird ein ausgewogener Skelettbaum mit der übergebenen Anzahl an Blättern aufgebaut.
Beide Konstruktoren greifen beim Baumaufbau auf die Methode **`insert()`** zu, die einen neuen Knoten in den Skelettbaum einfügt.

Das iterative Umsetzen der Anwendungsfälle zum Prioritätssuchbaum erfolgte in nachstehender Reihenfolge:

1. Einfügen von Punkten in einen semidynamischen Baum mit sofortiger Anzeige
2. Einfügen von Punkten in einen dynamischen Baum mit sofortiger Anzeige
3. Bereichssuche im semidynamischen Baum
4. Löschen von Punkten (semidynamisch und dynamisch)
5. Der semidynamisch statische Baum
6. Der balancierte dynamische Baum
7. Erweiterungen für die allgemeine Punktmenge
8. Animationen:
    - für das Einfügen von Punkten
    - für das Löschen von Punkten
    - für die Bereichssuche
9. Demos

Jeder Iterationsschritt endete mit gezielten Tests. Exemplarische Testbeispiele sind im Kapitel 9.2 beschrieben.

8.4.1 Einfügen von Punkten im semidynamischen Baum

Für das Einfügen von Punkten in den zeichenbaren Baum, haben wir den Einfügealgorithmus **`insertPoint()`** zunächst in der Klasse **`PrioritySearchTree`** aufgenommen und getestet. Nach erfolgreicher Implementation haben wir die Methode auch in die Klasse **`DrawablePrioTree`** übernommen und dort mit Zeicheninformationen und den Aufruf des Change-Listeners ergänzt.

Für die sofortige Ausgabe des geänderten Baumes in der Zeichenfläche muss ein Attribut im Knoten festhalten, ob ein Punkt neu eingefügt wurde. Wir haben dafür als zusätzliche Zeicheninformation das Attribut *`_contentsNew`* in die Klasse **`PaintInfo`** aufgenommen. Es wird auf *`true`* gesetzt, wenn ein neuer Punkt im Knoten abgespeichert wird.

In der Methode **`paint()`**der Baumausgabe-Klasse **`PrioTreeCanvas`** wird zunächst ein Preorder-Baumdurchlauf implementiert, um den Baum mit den abgespeicherten Punkten zu zeichnen. Unveränderte Punkte werden blau eingezeichnet, Punkte mit neuer Position rot.

Da die Adapterklasse **`InputAreaToTreeAndPointsAdapter`** bereits implementiert wurde (siehe Benutzeroberfläche in Kapitel 8.2.1) konnten wir im weiteren Verlauf der Softwareentwicklung alle Tests in grafischer Form durchführen

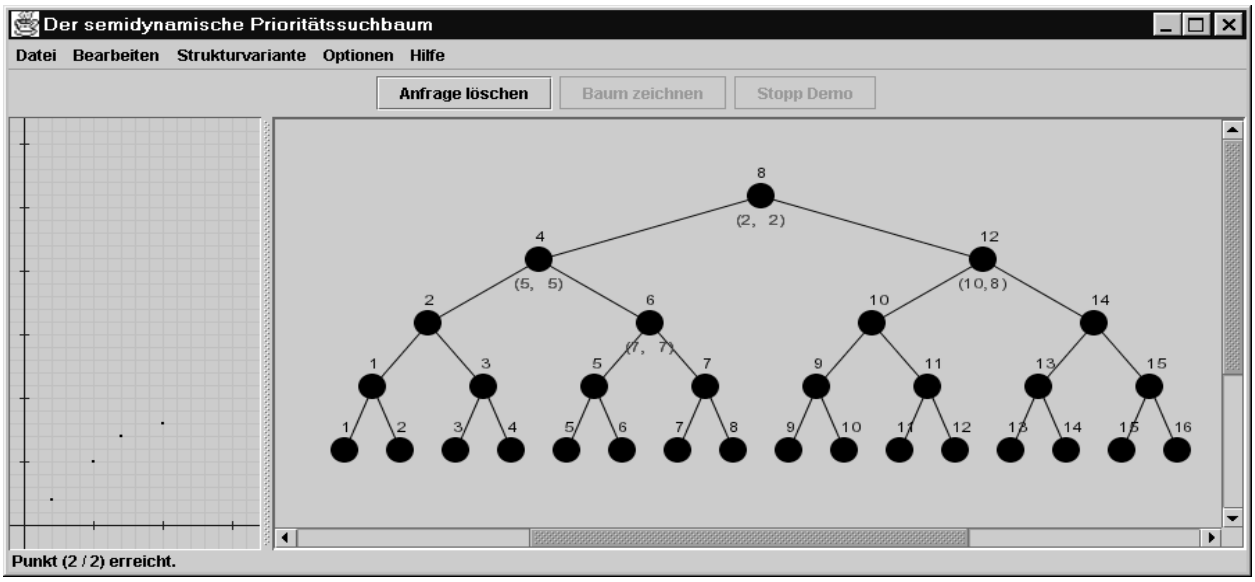


Abbildung 65: Einfügen von Punkten im semidynamischen Prioritätssuchbaum

Abbildung 65 zeigt den semidynamischen Baum nach Einfügen der Punkte (7|7), (5|5), (10|8) und (2|2). Der letzte Punkt (2|2) hat die Punkte (5|5) und (7|7) verdrängt. Der Punkt (10|8) hat seine Position nicht verändert.

Das Zusammenspiel aller am Einfügen eines Punktes beteiligten Objekte zeigen die Interaktionsdiagramme von Abbildung 66 und Abbildung 67.

Der **`InputAreaToTreeAndPointsAdapter`** wertet den MouseEvent aus und erzeugt den neuen Punkt mit den transformierten Werten der Bildschirmkoordinaten. Der Punkt wird dem Punktvektor hinzugefügt. Dieser informiert den **`PointVectorToViewAdapter`** über die Veränderung. Dieser wiederum veranlasst das Neuzeichnen der Punkte in der Eingabefläche (_view).

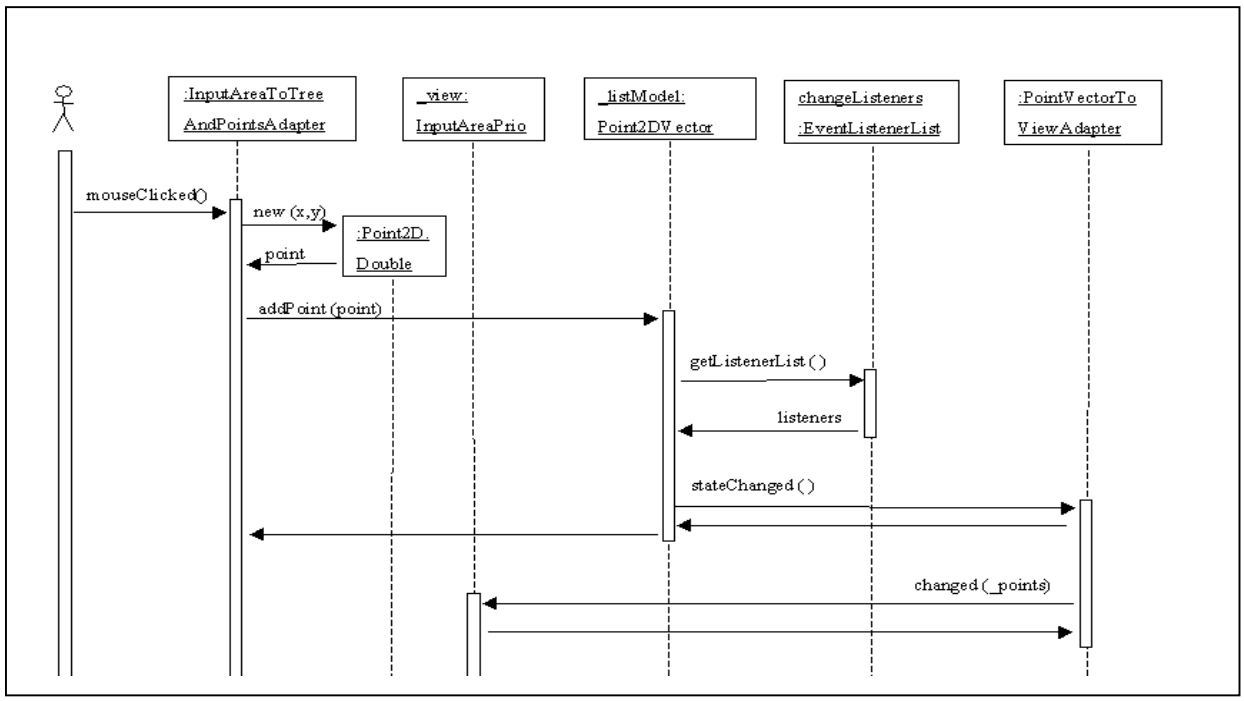


Abbildung 66: Interaktionsdiagramm: Einfügen im semidynamischen Baum (Teil 1)[80]

Der Punkt wird in den Baum eingefügt. Der Baum verständigt den **`PrioTreeToViewAdapter`** von der Änderung. Der Adapter legt eine neue Instanz der Klasse **`PrioTreeShow`** an, die diverse Methoden zum Zeichnen des Baums enthält. Die Zeichenmethoden werden nicht mehr in

[80] Zur Erläuterung der Symbole siehe Tabelle A11

der **`paint()`**-Methode von **`PrioTreeCanvas`** aufgerufen. Es wird jetzt in ein BufferedImage gezeichnet. Das Image wird an PrioTreeCanvas gesandt und dort nur noch von der **`paint()`**-Methode am Bildschirm eingeblendet.

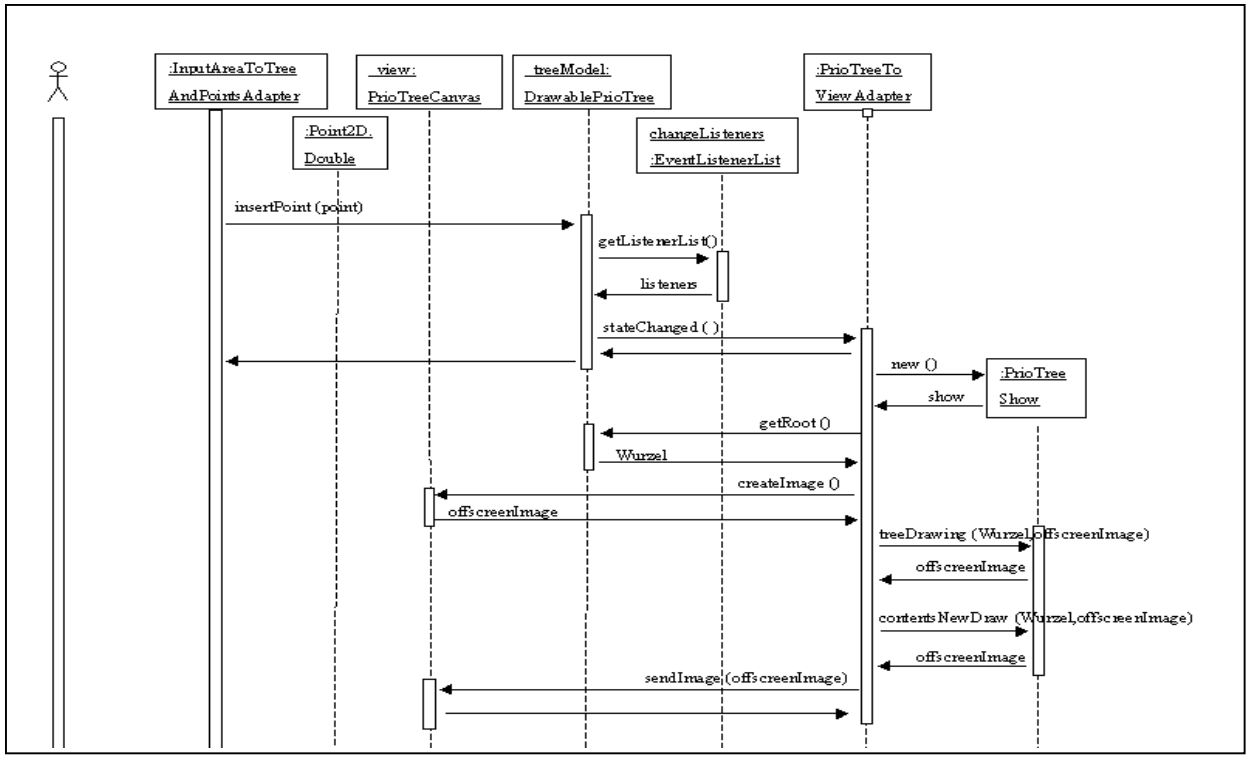


Abbildung 67: Interaktionsdiagramm: Einfügen im semidynamischen Baum (Teil 2)[81]

8.4.2 Einfügen von Punkten im dynamischen Baum

Für das Einfügen im dynamischen Fall muss die Methode **`insertPoint()`** in der Klasse **`DrawablePrioTree`** ergänzt werden. Im diesem Fall muss mit der Methode **`insert()`** zunächst der neue Knoten im Skelettbaum eingefügt werden, bevor die Methode zum Punkteinfügen **`pointsInSkeletonInsert()`** aufgerufen werden kann. Für die Fallunterscheidung wird in der Klasse **`DrawablePrioTree`** das Attribut *`treeType`* für den Baumtyp ergänzt.

[81] Zur Erläuterung der Symbole siehe Tabelle A11

8.4.3 Bereichssuche im semidynamischen Baum

Die Methode **`arraySearch()`** setzt bei ihrem Baumdurchlauf in allen besuchten Knoten das Flag *`_marked`* auf **`true`**. Die bei der Suche gefundenen Punkte werden mit *`_contentsNew = true`* festgehalten. Der am Ende der Methode aufgerufene ChangeListener veranlasst das Neuzeichnen des Baumes. Alle markierten Pfade, Knoten und Kanten, und die gefundenen Punkte werden grün eingezeichnet.

Abbildung 68 zeigt das Anfrageergebnis für einen Halbstreifen mit x-Intervall von 1 bis 7 und einem y-Grenzwert von 4. Zu untersuchen wären alle Knoten zwischen und auf den Pfaden zu Blatt 1 und Blatt 7. Da die Suche aber abgebrochen wird, sobald ein y-Wert auftritt, der größer ist als der Grenzwert 4 oder ein Knoten keinen Punkt enthält, kommen wir zu dem Suchverlauf von Abbildung 68.

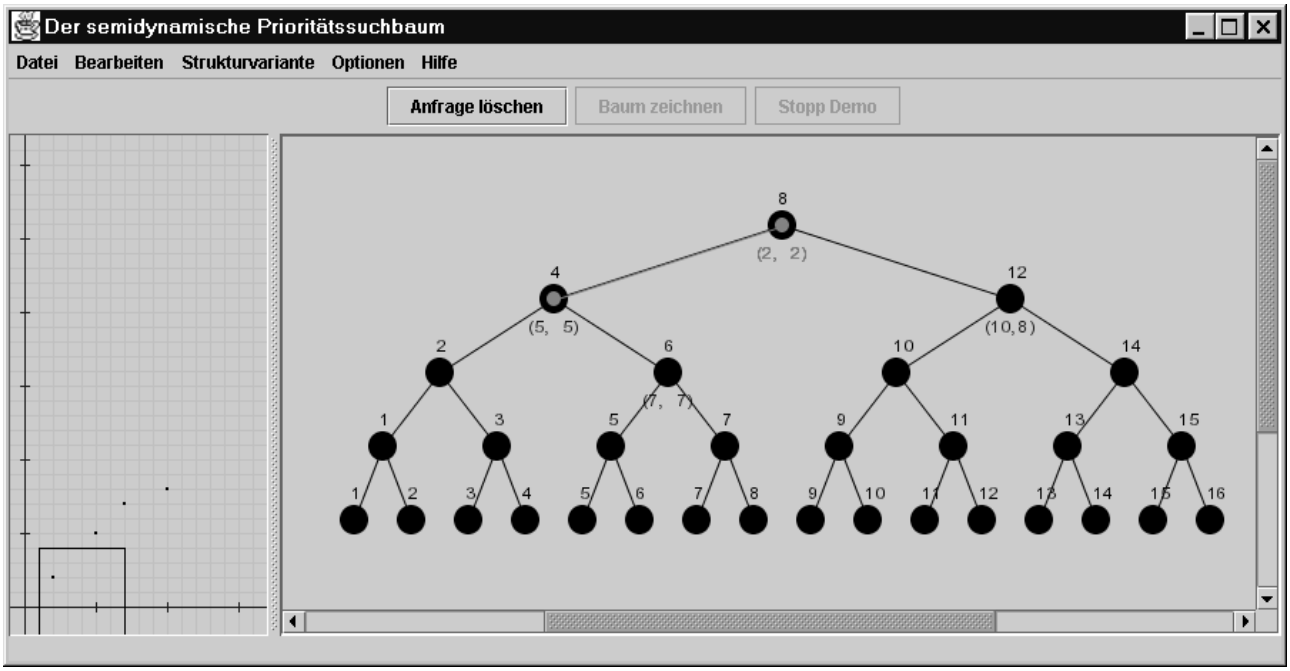


Abbildung 68: Bereichsanfrage im semidynamischen Prioritätssuchbaum

8.4.4 Löschen im semidynamischen und dynamischen Baum

Löschen im semidynamischen Prioritätssuchbaum

Nach gleicher Vorgehensweise wie beim Einfügen von Punkten wurde der Löschalgorithmus im semidynamischen Fall mit der Methode p**`ointFromSkeletonDelete()`** implementiert und getestet.

Löschen im dynamischen Prioritätssuchbaum

Im dynamischen Fall müssen zusätzlich zum Punkt auch die Knoten mit dem x-Wert des Punktes gelöscht werden. Hierfür haben wir die Methode **`deletePoint()`** ergänzt. Sie nutzt die Methode zum Löschen des Punktes im semidynamischen Fall und entfernt anschließend die zugehörigen Knoten mit der Methode **`delete()`**.

`delete()` sucht zunächst nach dem inneren Knoten.

Ist dessen linker Sohn der gesuchte Blattknoten, so ist das gesuchte Knotenpaar gefunden. Ansonsten muss der gesuchte Blattknoten den größten Splitwert im linken Teilbaum des inneren Knotens aufweisen.

Gelöscht werden immer der Blattknoten und sein Vorgängerknoten, auch wenn dies nicht der zugehörige innere Knoten ist. Der Vorgängerknoten kann jedoch noch einen Punkt gespeichert haben. Dieser muss vor dem Löschen durch einen Punkteinfügevorgang in den verbleibenden Nachfolgerknoten verlagert werden. Eingesetzt wird dazu die bereits fertige Methode **`pointInSkeletonInsert()`**.

Erst danach können die Knoten durch eine neue Verzeigerung gelöscht werden. Dabei wird der verbleibende Nachfolgerknoten zum Sohn des Vorgängers seines Vaterknotens.

War der innere Knoten nicht der direkte Vorgänger des gelöschten Blattknotens, muss der Splitwert des inneren Knotens mit dem des verbleibenden Nachfolgeknotens überschrieben werden. Abbildung 69 zeigt das Löschen des Punktes (4|5). Der rote Pfeil verdeutlicht die neue Verzeigerung. Der Splitwert des inneren Knotens (4) wird mit dem Wert 3 des verbleibenden Blattknotens überschrieben.

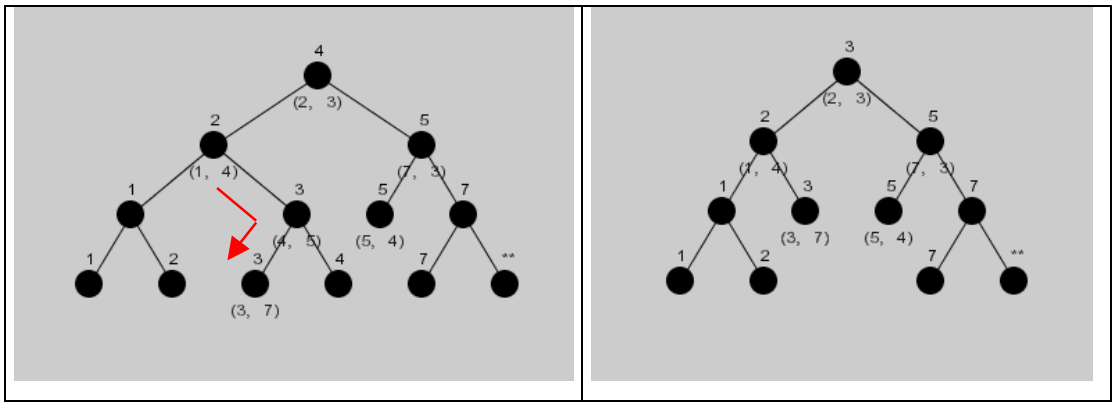


Abbildung 69: Löschen eines Punktes im dynamischen Prioritätssuchbaum

8.4.5 Der semidynamisch-statische Baum

Der Aufbau des statischen Baumes erfolgt erst nach der Eingabe aller Punkte. Der Baum soll ein ausgewogener Blattsuchbaum sein, der alle x-Werte der Punkte in den Blattknoten aufweist. Ausgewogen bedeutet, dass bis auf die unterste Ebene alle Ebenen vollständig besetzt sind. Es gibt nun verschiedene Möglichkeiten, einen solchen Baum zu erzeugen. Wir beschlossen, ihn linksvollständig darzustellen. Die Besetzung der untersten Ebene soll ohne freie Plätze zwischen den Blattknoten von links nach rechts zunehmen. Dies ergibt ein regelmäßiges Aussehen der Bäume (siehe Abbildung 70).

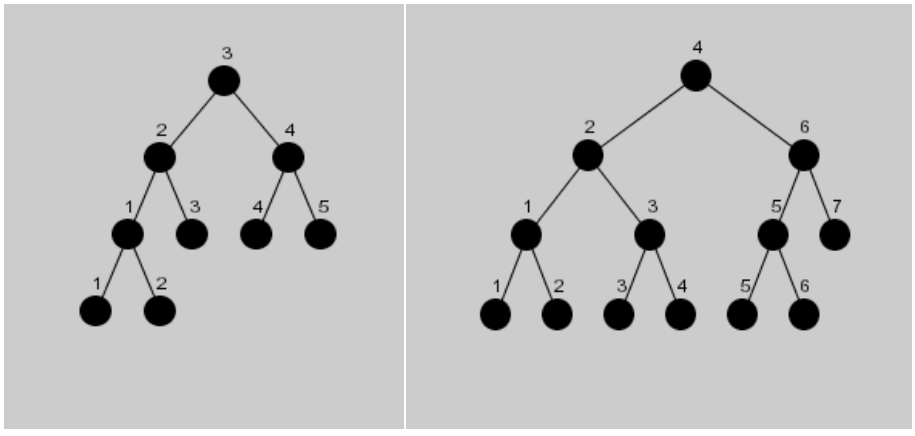


Abbildung 70: Linksvollständige Baumskelette

Für den statischen Skelettbaum entwerfen wir einen weiteren Konstruktor in der Klasse **`DrawablePrioTree`**, an den eine nach x aufsteigend sortierte Punktliste übergeben wird. Der Konstruktor ruft die Methode **`_staticTree2()`** auf, die folgende Algorithmus-Idee implementiert.

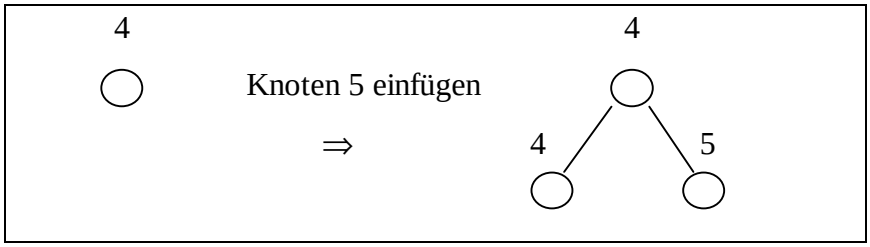
Ist die Knotenanzahl eine 2erPotenz, dann wird durch 2 geteilt, um den Wert der Variablen MitteIndex zu ermitteln. Sonst:

1. Nächstgrößere 2erPotenz ermitteln
2. Nächstkleinere 2erPotenz ermitteln
3. Differenz zwischen den 2erPotenzen bilden
4. Knotenanzahl < (kleine 2erPotenz + Differenz/2)

   Dann MitteIndex = kleine 2erPotenz/2 + (Knotenanzahl – kleine 2erPotenz)

   Sonst MitteIndex = große 2erPotenz/2
5. MitteIndex = von + MitteIndex - 1
6. Mit MitteIndex auf die Punktliste zugreifen, um den x-Wert des Punktes zu ermitteln.
7. Neuen Knoten mit x-Wert in den Skelettbaum einfügen.
8. **`_staticTree2(`***`Mitte+1, bis`***`)`**
9. **`_staticTree2(`***`von, Mitte-1`***`)`**

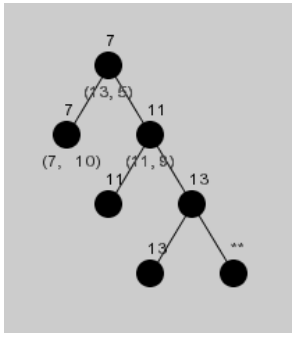
Der rekursive Aufruf von **`_staticTree2()`** muss nach rechts beginnen, damit nach den Regeln für den Blattsuchbaum der zuvor ermittelte Knoten seine Position behält.
Nach links muss die Knotenanzahl stets um 1 erhöht werden, weil der Knoten mit dem kleineren Wert als linker Sohn ein zweites Mal eingefügt wird.



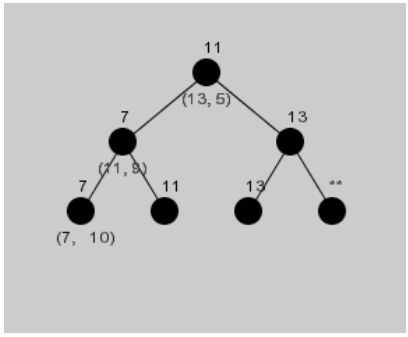
Das Einfügen der Punkte in den Skelettbaum erfolgt wie im semidynamischen Fall mit der Methode **`pointsInSkeletonInsert()`**. Nur sind im statischen Fall schon alle Punkte bekannt, deshalb werden sie vor dem Einfügen nach den y-Koordinaten sortiert. So kommt es zu keinen Punktverdrängungsvorgängen.

8.4.6 Der balancierte dynamische Baum

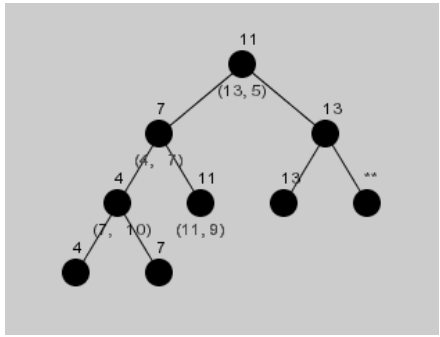
Der balancierte Fall setzt auf den Methoden des dynamischen Falls auf. Wird ein neuer Punkt mit dem zugehörigen Knoten eingefügt, entsteht zunächst ein eventuell unbalancierter Baum. Anschließend wird der neu eingefügte Knoten erneut aufgesucht, um den notwendigen Balancierungsvorgang durchzuführen.



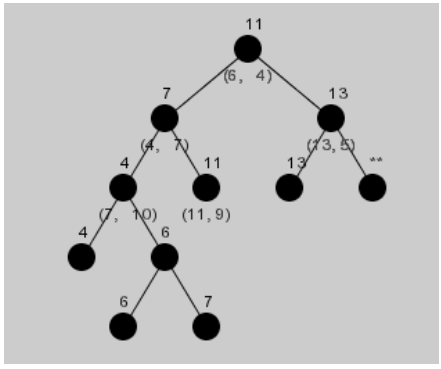
(7/10), (11/9), (13/5)



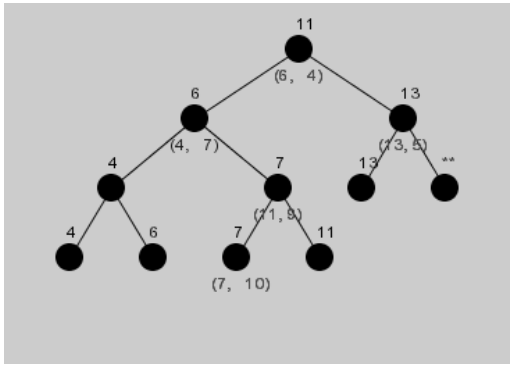
→ Rotation nach links



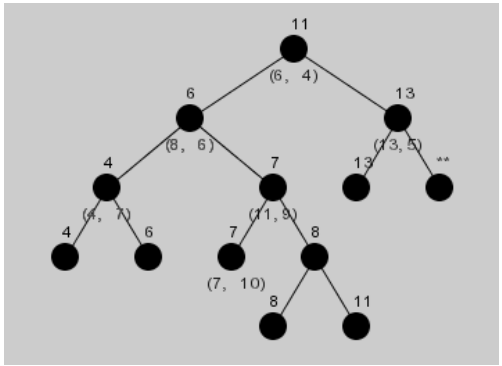
(4/7), Baum bleibt balanciert



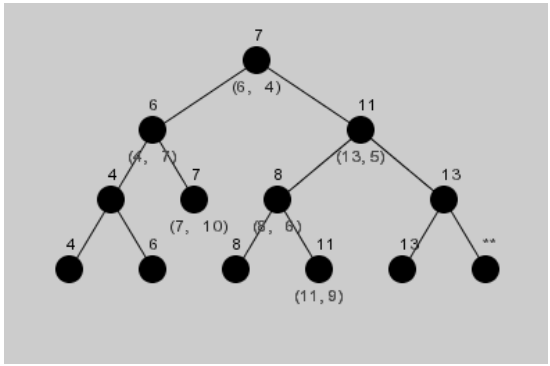
(6|4) eingefügt



→ Doppelrotation links/rechts



(8|6) eingefügt



→ Doppelrotation links/rechts

Abbildung 71: Beispiel zum Punkteinfügen in den dynamisch balancierten Baum

Dafür haben wir als Basis den für AVL-Bäume üblichen Balancierungsalgorithmus mit einfacher oder Doppelrotation eingesetzt. Der Algorithmus musste natürlich an die Gegebenheiten im Prioritätssuchbaum angepasst werden. Das Rotieren darf den y-Heap der im Baum gespeicherten Punkte nicht verletzen. Dazu muss vor einer Rotation der zukünftige Wurzelknoten des rotierenden Teilbaumes freigemacht werden. Ein dort abgespeicherter Punkt muss auf seinem Pfad zum Blatt eine Position tiefer neu eingefügt werden. Nach der Rotation wird die entstandene Lücke im neuen Wurzelknoten durch „Hochziehen“ des passenden Punktes geschlossen. Im Beispiel in Abbildung 71 wird die Punktfolge (7|10), (11|9), (13|5), (4|7), (6|4) und (8|6) eingegeben. Nach dem Einfügen des Punktes (13|5) reicht eine einfache Rotation. Nach (6|4) und (8|6) sind Doppelrotationen erforderlich, um die Balance wiederherzustellen. Zum Aufrechterhalten des y-Heaps dienen die fertigen Punkteinfüge- und Löschmethoden, die wir in die Balancierungsmethoden integriert haben.

Löschen dynamisch balanciert

Das Löschen von Punkt und Knoten erfolgt mit denselben Methoden wie im dynamischen Fall. Ist der Baum nach dem Löschen außer Balance, so wird nicht sofort beim Auflösen der Rekursion balanciert, sondern wie schon beim Einfügen eines Punktes im dynamisch balancierten Fall wird der Balancierungsvorgang vom Löschvorgang getrennt. Dies ist für das spätere animierte Zeichnen noch wichtiger als beim Einfügen, da das Balancieren nach dem Löschen eines Knotens mehrfach notwendig werden kann.

8.4.7 Erweiterung für die allgemeine Punktmenge

Entfällt die Einschränkung, dass nur Punkte mit disjunkten x-Werten erlaubt sind, können beliebig viele Punkte mit gleicher x-Koordinate auftreten. Damit ist nicht mehr garantiert, dass die Punkte auf ihrem Einfügepfad einen freien Speicherplatz vorfinden. Deshalb haben wir, wie in [MEH94b] vorgeschlagen, den Blattknoten um eine Priority-Queue erweitert, die diese „überzähligen“ Punkte aufnimmt. Als Implementierung haben wir eine einfache sortierte Punktliste eingesetzt, da in unserem Programm nicht mit großen Punktmengen gearbeitet wird. Ansonsten wäre eine balancierte Baumstruktur für die Implementierung vorzuziehen.

Da die Punkte bereits im disjunkten Fall als ((x,y),y) mit lexikografischer Ordnung der ersten Komponente bei gleichen y-Werten eingefügt werden, benötigen wir für das Einfügen und Löschen im semidynamischen Fall nur die Anpassung der vorhandenen Methoden **`pointInSkeletonInsert()`** und **`pointFromSkeletonDelete()`** für das Verhalten im Blattknoten.

- Einfügen: Ist der Blattknoten besetzt, muss in die Liste eingefügt werden.
- Löschen: Ist im Blatt der Punkt nicht gefunden, muss in der Liste gelöscht werden.
  Wird der Blattknoten frei, muss der erste Wert der Liste in den Blattknoten übernommen und in der Liste gelöscht werden.

Im dynamischen Fall ist zusätzlich zu berücksichtigen, ob im Ansatzknoten (Blatt) für einen neu einzufügenden Knoten eine nichtleere Liste existiert. Ist dies der Fall, so muss die Liste in den zugehörigen Nachfolgeknoten umgehängt werden, nachdem der erste Listeneintrag gelöscht und dieser Punkt in den neuen Blattknoten eingefügt wurde. Die benötigten Änderungen wurden in der Methode **`insert()`** vorgenommen.

Da die benötigte sortierte Punktliste den Knoten zuzurechnen ist, haben wir sie als weiteres Attribut zusammen mit den notwendigen Methoden in der Klasse **`PrioNode`** ergänzt.

8.4.8 Animationen zum Prioritätssuchbaum

Die Animation der Abläufe in den Methoden zum Einfügen, Löschen und der Bereichsanfrage muss durch zusätzliche Threads erfolgen, damit der Benutzer die Option behält, während der Animation auf die Benutzeroberfläche zuzugreifen und sei es nur, um die Animation zu stoppen.

Da das Visualisieren der Algorithmen die vorgenommenen Aktionen gut nachvollziehbar aufzeigen soll, müssen diverse Zwischenergebnisse beim Abarbeiten der Algorithmen dargestellt werden. Wir haben uns entschieden, diese Informationen in zusätzlichen „Animations-Datenstrukturen" festzuhalten, während der „normale" Algorithmus abläuft.

Wir konnten dazu die vorhandenen Algorithmen ergänzen. In unterschiedlichen Listen werden Informationen über den alten Baum, über Suchpfade, Einfügevorgänge für Knoten oder Löschpfade in der Klasse **`DrawablePrioTree`** festgehalten.
Die Listen werden den Animationsobjekten über Methoden vom Baumobjekt zur Verfügung gestellt.

Die Zeicheninformation wird begleitend zu den rekursiven Aufrufen in den Baumalgorithmen zusammengestellt. Sie enthält jeweils Start- und Zielknoteninformation und macht so für die Animation die eingeschlagenen Wege verfügbar.
Jede einzelne Information umfasst Knotenkoordinaten, Split- und Punktwerte.

Die Zeichenthreads der Animation nutzen diese Informationen zum schrittweisen Aufbereiten der Abläufe beim Einfügen, Löschen und der Bereichsanfrage, obwohl der Baum bereits komplett in der fertigen neuen Struktur vorliegt.

Für das animierte Zeichnen ist der alte Baum in modifizierter Weise festzuhalten. Er muss für den Betrachter die alten Inhalte aufweisen, für die kommenden Darstellungen aber bereits mit den aktualisierten Zeichenkoordinaten dargestellt werden, wie Abbildung 72 verdeutlicht.

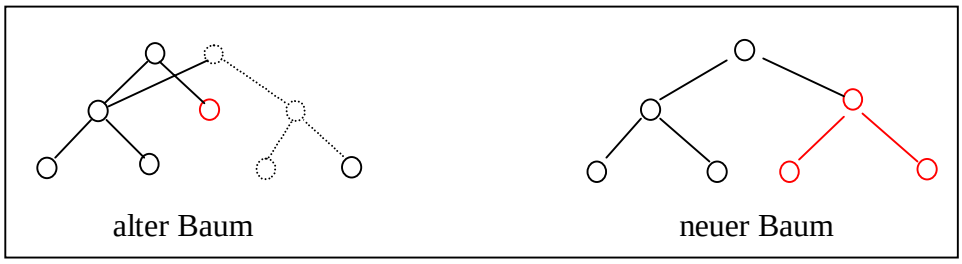


Abbildung 72: Animierte Erweiterung des alten Baums

Die Offsets und Zeichenkoordinaten für den Baum müssen also korrigiert sein, bevor das Einfügen als Film ablaufen kann. Ohne die Korrektur ist kein Platz für die neuen Blattknoten vorhanden, sie würden alte Knoten überzeichnen. Das bedeutet, dass wir die Information für den alten Baum aus dem neuen Baum gewinnen, in die Infoliste alle Knoten bis auf die neu eingefügten übernehmen und zusätzlich für den alten Blattknoten dessen alten Splitwert zur Verfügung stellen müssen.

Die verschiedenen Informationslisten bilden die Grundlage für die Animation. Diese wird hier für das animierte Knoteneinfügen in fünf Teilschritten dargestellt.

1. Alten Baum mit aktualisierten Zeichenkoordinaten aufbauen
2. Suchpfad zum Ansatzknoten einzeichnen
3. Linken Knoten entwickeln
4. Splitwerte aktualisieren
5. Rechten Knoten entwickeln

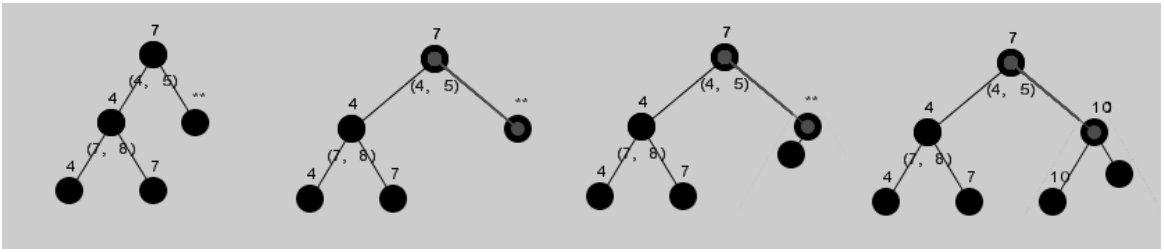


Abbildung 73: Animiertes Einfügen eines neuen Knotens

Das Einfügen der neuen Punkte in den Baum erfolgt beim semidynamischen und beim dynamischen Fall durch denselben Zeichenthread, nachdem im dynamischen Fall zuvor das Baumgerüst um den neuen Knoten erweitert wurde, wie in Abbildung 73 verdeutlicht wird.

Wie in Kapitel 7.5.3.2 beschrieben soll der einzufügende Punkt dabei langsam auf seinem Einfügepfad in den Baum hineinwandern. Wird durch ihn ein anderer Punkt verdrängt, sollen beide Punkte den Platz tauschen. Der verdrängte Punkt muss weiter in den Baum hineinlaufen. Die in der Methode **`pointInSkeletonInsert()`** erstellte Informationsliste enthält dafür in den Einzeleinträgen den Punkt im aktuellen Knoten, den neuen Punkt und ob getauscht wird oder nicht. Aus den Knoteninformationen berechnen Methoden aus der Klasse **`AnimInfo`** die Anzahl der Zwischenschritte und die temporären Positionen für das Gleiten der Punkte. Pro Zwischenschritt wird im Animation-Thread ein Image gezeichnet, das dann zur Anzeige an die Klasse **`PrioTreeCanvas`** gesandt wird. Diese blendet nur noch das gesamte Image mit der **`paint()`**-Methode in die Baumzeichenfläche ein. Beispiele sind in Abbildung 74 und Abbildung 75 zu sehen.

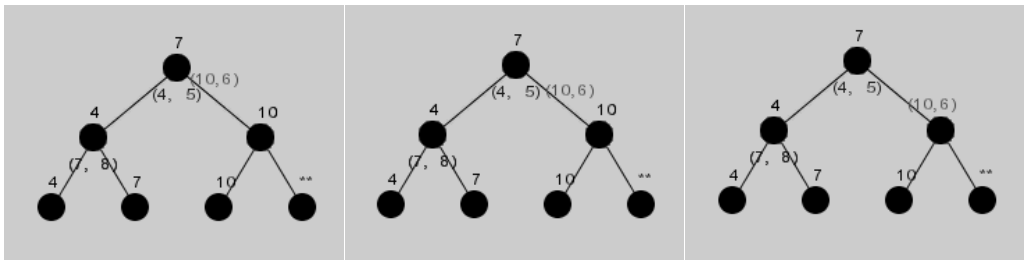


Abbildung 74: Animiertes Einfügen eines neuen Punktes

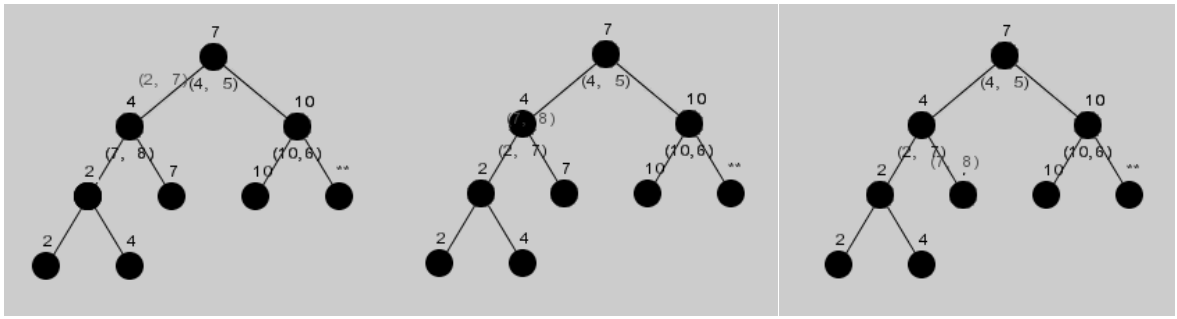


Abbildung 75: Animiertes Einfügen mit Verdrängen eines Punktes

Beim animierten Löschen im dynamischen Fall, siehe Abbildung 76, sind wir in unserem Drehbuch den Überlegungen aus Kapitel 7.5.3.4 gefolgt:

1. Es wird ein Löschpfad zu dem Knoten eingezeichnet, der den zu löschenden Punkt enthält.
2. Der Punkt wird gelöscht, der Löschpfad entfernt, die Lücke animiert geschlossen.
3. Der Suchpfad zum zu löschenden Knoten wird eingezeichnet. Dieser endet in dem Vaterknoten des zu löschenden Blattknotens.
4. Ist noch ein Punkt im Vaterknoten abgelegt, so muss dieser animiert tiefer im Baum neu eingefügt werden.
5. Dann wird die Kante rot eingezeichnet, um das Zeigerumhängen zu verdeutlichen. Alle Knoten stehen noch an ihren alten Positionen, aber die gelöschten sind ausgeblendet.
6. Abschließend wird der neue Baum mit den korrigierten Knotenpositionen angezeigt.

Die Informationslisten für alle Zwischenschritte werden von den Löschmethoden im **`DrawablePrioTree`** aufbereitet. Die Löschanimation arbeitet alle Zwischenschritte ähnlich wie beim Punkteinfügen sequentiell ab, sendet jedes neuaufbereitete Image zum Anzeigen an die Baumzeichenfläche. Nur wandern die Punkte beim Löschen vom Zielknoten zum Startknoten rückwärts den Baum hinauf.

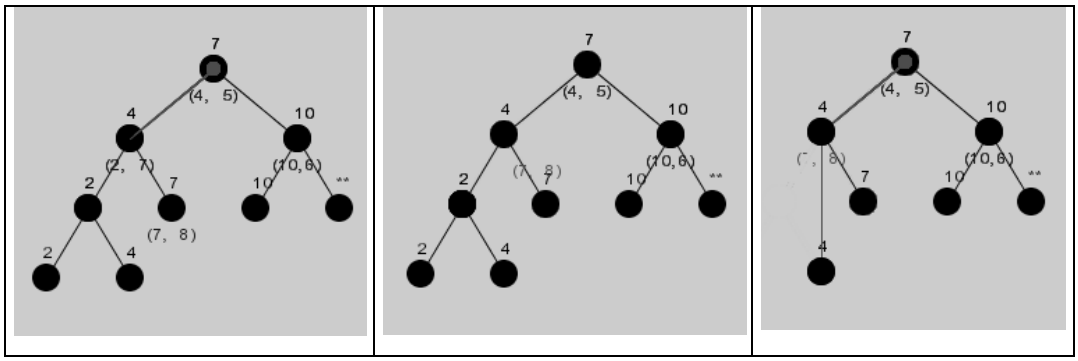


Abbildung 76: Animiertes Löschen im dynamischen Prioritätssuchbaum

Beim Erstellen der Animationen gab es vor allem Probleme mit der Ablaufgeschwindigkeit. Anfangs war es auch schwierig, zu einer ruhigen, flackerfreien Darstellung zu gelangen. Zuerst haben wir für das Einfügen von Punkten in den dynamischen Prioritätssuchbaum mit drei aufeinanderfolgenden Threads gearbeitet, die zunächst den alten Baum gezeichnet, dann die Knoten ergänzt und zuletzt den Punkt eingefügt haben. Sie wurden in der **`paint()`**-Methode der Klasse **`PrioTreeCanvas`** aufgerufen.

Da das Umsetzen der oben beschriebenen Vorgehensweise leider nur unter Windows zu vertretbaren Ergebnissen geführt hat, aber leider nicht unter Linux und Solaris, musste dieser Weg aufgegeben werden.

Als neuen Ansatz wählten wir das Zeichnen in ein BufferedImage, ein Hintergrundbild, wie es das Java-Paket **`Graphics2D`** zur Verfügung stellt. Es gestattet das Vorbereiten einer komplexen Darstellung im Hintergrund, während noch ein anderes Bild am Monitor zu sehen ist. Das fertige Image wird dann nur noch in den Anzeigebereich kopiert. Dies gestattet ein schnelles Anzeigen eines kompletten Bildes in der **`paint()`**-Methode der View-Klasse **`PrioTreeCanvas`**.

Zunächst wurde pro Bild der gesamte alte Baum neu gezeichnet, bevor der wandernde Punkt an seiner neuen Position eingezeichnet wurde. Dies führte zu einer sauberen flimmerfreien Darstellung der Abläufe, aber die Geschwindigkeit der Animation war vor allem unter Unix wieder zu langsam.

Erst das Einschränken des Neuzeichnens auf überschaubar kleine Bereiche brachte eine Besserung, so dass jetzt die Geschwindigkeit unter allen drei Betriebssystemen vertretbar ist.

Leider hat dadurch aber manchmal die „Sauberkeit“ gelitten. Es werden immer noch Bereiche gelöscht, die nicht wieder sofort nachgezeichnet werden. Dies fällt besonders auf, wenn zwei unterschiedliche Äste im Baum sehr dicht nebeneinander liegen.

8.4.9 Demos zum Prioritätssuchbaum

Die zur Verfügung gestellten Demos werden als separate Thread-Klassen realisiert. Sie werden über Unterpunkte des Hilfemenüs aufgerufen und laufen automatisch ab, bis die Threads beendet sind oder der Benutzer den Ablauf unterbricht.
Jedes Demo ist eine Sequenz von Punkteinfüge- und Punktlöschvorgängen und einer abschließenden Bereichsanfrage.
Nachdem ein Feld mit Punkten initialisiert ist, werden in einer Demoklasse die gewünschten Aktivitäten sequentiell aufgerufen. Alle dafür benötigten Methoden stehen bereits zur Verfügung.

Ein Demo kann vom Benutzer nur wohldefiniert, d.h. mit dem Ende einer Transaktion abgebrochen werden. Stoppt der Benutzer das Demo über die Stopp-Schaltfläche, wird die aktuelle Aktion noch zu Ende geführt. Erst dann erfolgt der Abbruch des Demos.
Das Beenden der Demo-Threads geschieht über eine boolesche Variable *_continue*, die beim Start des Demos auf *true* und durch die vom Benutzer aufgerufene **stop()**-Methode des Demos auf *false* gesetzt wird. Diese Vorgehensweise ersetzt die deprecated Threadmethode **Thread.stop()**. Diese Methode soll nicht mehr verwendet werden, weil Transaktionen dadurch willkürlich unterbrochen wurden.

Auch die Methoden **suspend()** und **resume()** gelten als deprecated und werden deshalb in unserem Programm nicht mehr eingesetzt. Das Zusammenspiel von Threads regeln wir ebenfalls über boolesche Variablen. Ruft das Demo einen weiteren Thread, z.B. zum Punkteinfügen auf, so wird zuvor vom Demo die boolesche Variable *more* in der Klasse **PrioTreeGUI** auf *false* gesetzt. Dann wartet der Demothread, bis der aufgerufene Einfügethread beendet und dadurch die Variable *more* in **PrioTreeGUI** auf *true* gesetzt ist. Dies Zusammenspiel der Threads zeigt auch das Interaktionsdiagramm von Abbildung 77.

Erst nach dem Ende des Demos stehen dem Benutzer wieder alle Menüpunkte zur Verfügung. Während ein Demo abläuft, darf kein weiteres gestartet werden. Beide würden nebenläufig arbeiten, ihre Ausgaben gleichzeitig in der Baumzeichenfläche darstellen und so für ein heilloses Durcheinander sorgen.

Die Demoklassen werden auch genutzt, wenn der Benutzer in der Option „animierte Darstellung“ ein Neuzeichnen des Baumes verlangt. Nur wird hier das Demo statt mit den vorbereiteten Demowerten mit der aktuellen Punktliste abgearbeitet.

Das „Demo“ ist in diesem Fall aber nur ein kontinuierliches Einfügen der aktuellen Punkte, es kann jederzeit durch eine andere Benutzeraktion abgebrochen werden, die Menüpunkte bleiben weiter präsent.

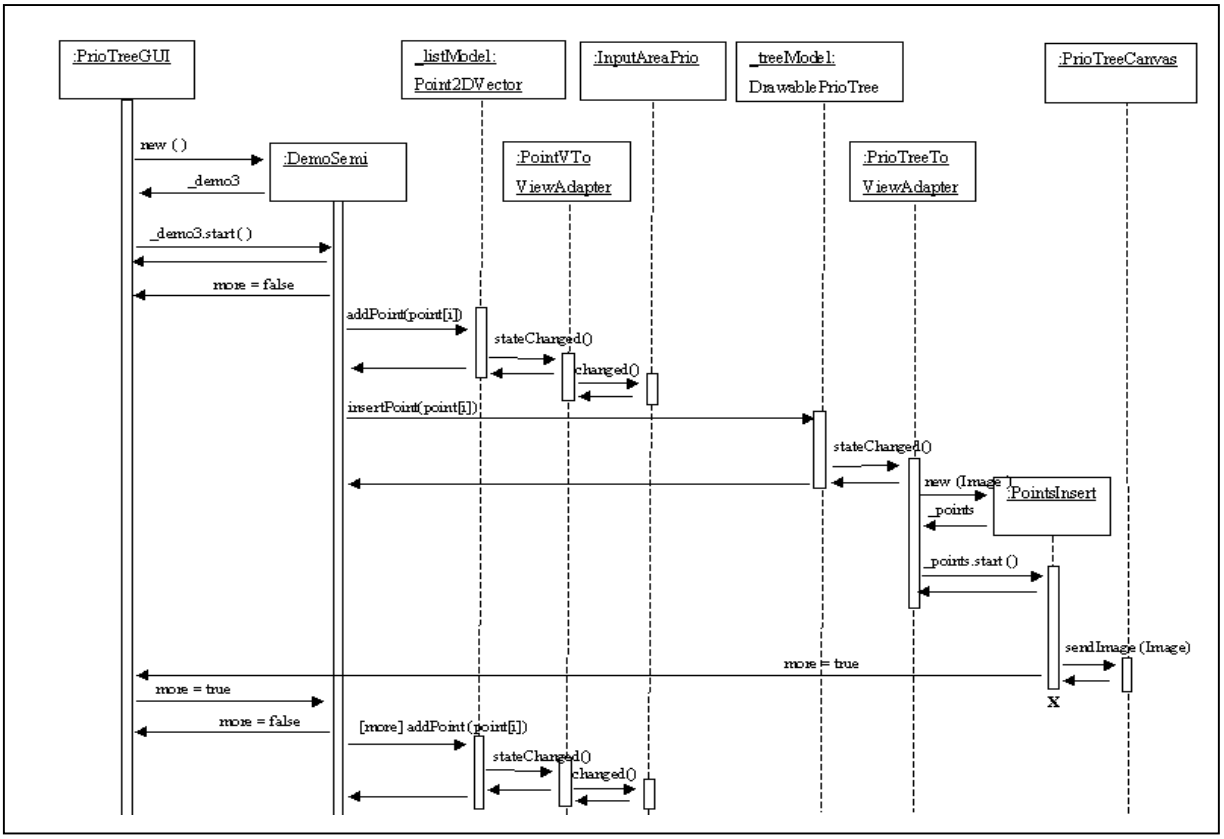


Abbildung 77: Interaktionsdiagramm zum „Zusammenspiel der Threads“[82]

Erläuterungen zum Interaktionsdiagramm von Abbildung 77

Der zunächst gestartete Demo-Thread **`_demo3`** setzt die Variable *`more`* in :**`PrioTreeGUI`** auf *`false`*, ruft dann das Einfügen des ersten Punktes in die Punktliste und den Baum auf. Am Ende der Modell-Änderungen werden die ChangeListener verständigt.

[82] Zur Erläuterung der Symbole siehe Tabelle A11

Das Zeichnen des Punkteinfügens in den Baum geschieht durch den Thread **`_points`**. Der ChangeListener übergibt ihm dafür ein BufferedImage der Zeichenfläche. Jedes veränderte Bild schickt der Zeichenthread zum Anzeigen an die Instanz von **`PrioTreeCanvas`**. Am Ende der **`run()`**-Methode setzt **`_points`** die Variable *`more`* auf *`true`*, damit der nächste Einfügevorgang beginnen kann. Nach Abschluss der **`run()`**-Methode wird die Zeichenthread-instanz automatisch gelöscht. Der Demo-Thread **`_demo3`** hat gewartet bis *`more`* den Wert *`true`* aufweist. Jetzt kann er mit dem Einfügen des nächsten Punktes fortfahren.

9. Programmtest

Nach dem Rational Unified Process endet jede Iteration mit einer Ergebnisprüfung, dem Review. Wir müssen also Testsituationen, Testfälle und Testdaten aufstellen, mit denen sich das jeweilige Iterationsziel möglichst genau überprüfen lässt.

Um den Anforderungen der analytischen Qualitätssicherung zumindest ansatzweise gerecht zu werden, haben wir zur Validierung von **`VisualGeom`** folgende Testverfahren eingesetzt:

- White-Box-Tests, um Sequenzen, Auswahlen und Wiederholungen im Rahmen bestimmter Methoden zu testen
- Black-Box-Tests, um in hinreichendem Umfang die Funktionalität von **`VisualGeom`** zu gewährleisten
- Code-Review zum Auffinden besonders „hartnäckiger“ semantischer Fehler

Beim Auftreten von Fehlern, z. B. dem Abweichen des beobachteten Zustandes der Graphen vom theoretisch zu erwartenden Bild, haben wir die jeweilige Fehlerursache durch die Ausgabe von Zwischenwerten oder das gezielte Debuggen mit Breakpoints und dem Verfolgen von Variablenzustände eingekreist. Anschließend wurde die jeweilige Fehlerursache beseitigt.
Hilfreich war dabei die von uns genutzte integrierte Entwicklungsumgebung[83]. Sie ermöglicht im Gegensatz zum im JDK integrierten Debugger eine komfortable Fehlersuche.

Kontrolliert werden muss allerdings noch, ob unser Softwareprodukt den aufgestellten ergonomischen Anforderungen genügt. Dies bedeutet u. a. eine Evaluation bei den Nutzern der Software. Aus der Menge möglicher Evaluationsmethoden [FU4763, Seite 91] sehen wir bei dem Adressatenkreis folgende Möglichkeiten:

- In Anlehnung an [FU 4763 Seite 92 bis 94] sollte zunächst ein Fragebogen ausgearbeitet werden und damit eine schriftliche Befragung der Studenten im Zusammenhang mit der Bearbeitung des Kurstextes [FU1840] erfolgen.

[83] Kawa von der Firma Tek-Tools, allerdings nur unter Windows lauffähig

- Auch eine Online-Befragung bei der Nutzung von **`VisualGeom`** als Applet in der virtuellen Universität wäre denkbar.
- Die Evaluation findet entlang von Standardaufgaben statt, die als Einsendeaufgaben konzipiert werden und unter Anwendung von **`VisualGeom`** zu lösen sind.

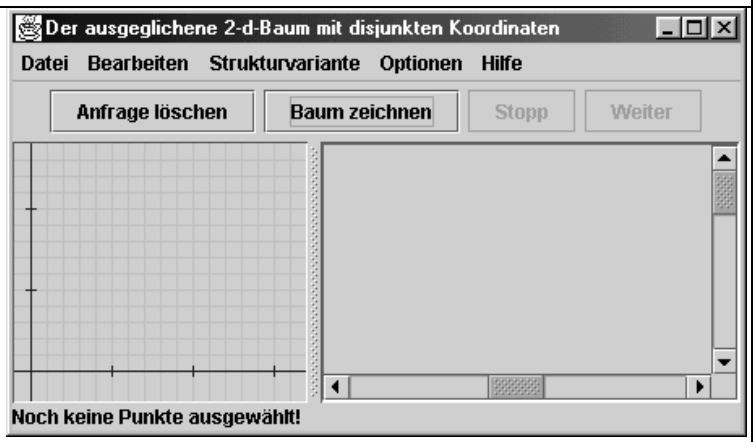
Diese Evaluation können wir jedoch nicht im zeitlichen Rahmen unserer Diplomarbeit durchführen.

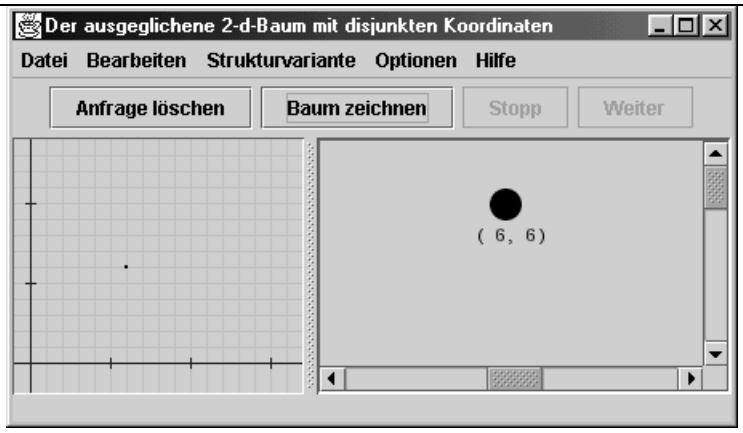
Exemplarisch werden nachfolgend an ausgewählten Beispielen zum 2-d-Baum und zum Prioritätssuchbaum einige der von uns durchgeführten Tests beschrieben.

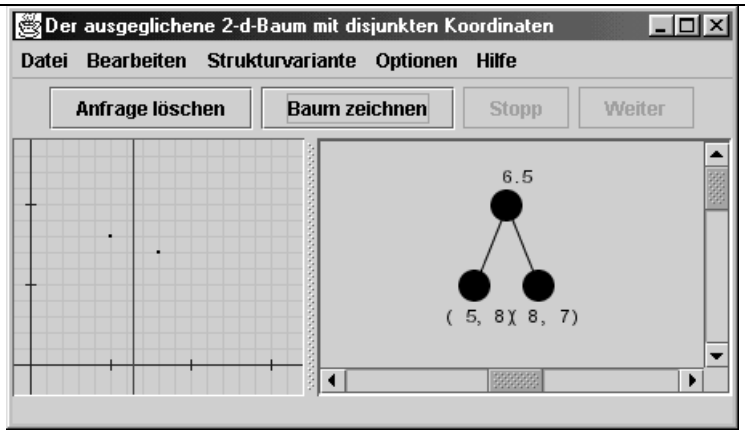
9.1 Tests zum 2-d-Baum

Beispiele für den Test spezieller Werte

Typische Beispiele hierfür sind das Verhalten der Software bei dem Versuch, den leeren, den aus einem Punkt bestehenden oder den aus zwei Punkten bestehenden 2-d-Baum aufzubauen oder das Speichern und Laden einer leeren Punktliste durchzuführen. Diese Tests werden nachfolgend für den ausgeglichenen 2-dimensionalen Suchbaum dargestellt.

Erwartetes Ergebnis	Verhalten des Programms
Beim Betätigen des Buttons „Baum zeichnen“ muss in der Statuszeile der Hinweis ausgegeben werden, dass noch keine Punktliste existiert und somit kein 2-d-Baum gezeichnet werden kann.	 **Beispiel 1: Test des Aufbaus eines leeren Baums**

Erwartetes Ergebnis	Verhalten des Programms
Beim Betätigen des Buttons „Baum zeichnen“ muss der Wurzelknoten als Blatt im Ausgabebereich erscheinen und an ihm der Punktwert ausgegeben werden. Es existiert noch keine Splitgerade.	 **Beispiel 2: Test des Aufbaus des Baums mit einem Knoten**

Erwartetes Ergebnis	Verhalten des Programms
Beim Betätigen des Buttons „Baum zeichnen“ muss der Splitwert als Mittelwert der beiden x-Koordinaten berechnet, der Knoten mit dem Splitwert und die beiden Blätter mit den Punktwerten im Ausgabebereich dargestellt werden. Im Eingabebereich ist mit dem Splitwert eine senkrechte Gerade einzuzeichnen.	 **Beispiel 3: Test des Aufbaus des Baums mit zwei Knoten**

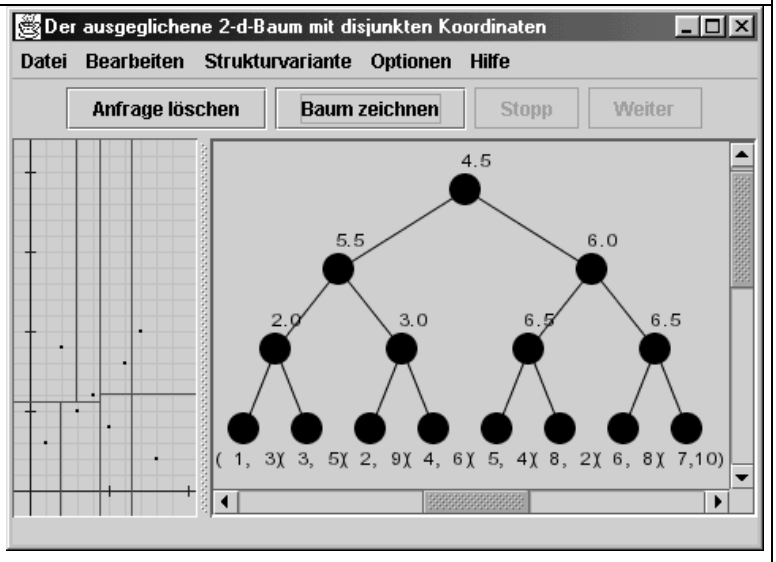
Erwartetes Ergebnis	Verhalten des Programms
Auch nach dem Speichern einer leeren Punktliste soll das Laden ohne Fehlermeldung möglich sein. Dem Benutzer ist mitzuteilen, dass die geladene Punktliste leer ist und deshalb kein Baumaufbau erfolgt.	**Beispiel 4: Laden einer leeren Punktliste**

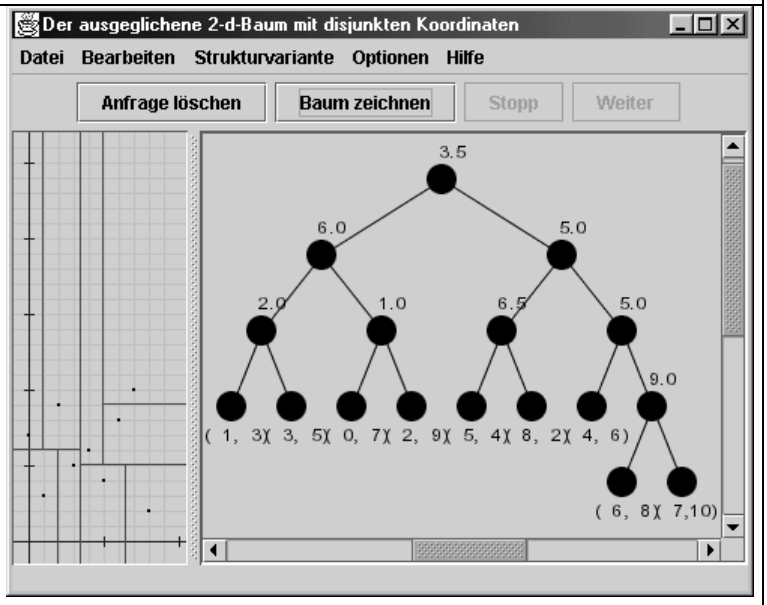
Beispiele für das Testen funktionaler Äquivalenzklassen

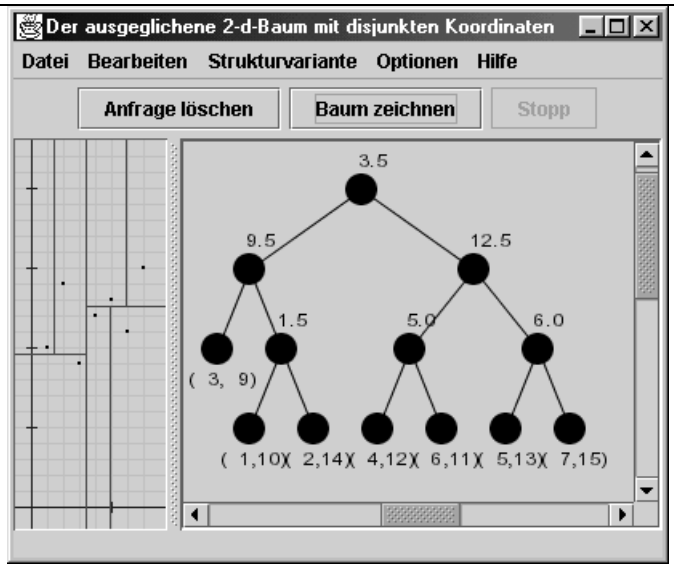
Für den Test des ausgeglichenen 2-dimensionalen Suchbaums wird eine grobe Einteilung in Eingabe- und Ausgabeäquivalenzklassen vorgenommen. Mit Stichproben aus den jeweiligen Äquivalenzklassen werden die folgenden Testbeispiele ausgeführt.

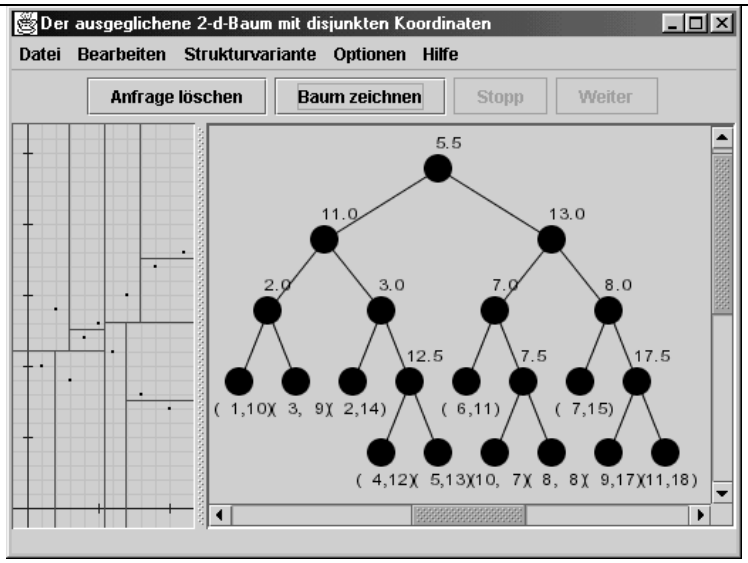
Wir testen das Verhalten beim Baumaufbau mit folgenden Ausgabeäquivalenzklassen:

- leerer Baum (s. Beispiel 1)
- Baum mit nur einem Blatt (s. Beispiel 2)
- Baum mit einem Splitknoten und zwei Blättern (s. Beispiel 3)
- Baum, der in der untersten Ebene die maximal mögliche Anzahl von Blättern besitzt (s. Beispiel 5) und mehr als 2 Ebenen hat
- Baum, der in der untersten Ebene nicht die maximal mögliche Anzahl von Blättern besitzt
  a) Baum, der in der untersten Ebene nur zwei Blätter besitzt (Beispiel 6)
  b) Baum, der in der untersten Ebene zwei Blätter weniger als die maximal mögliche Anzahl besitzt (Beispiel 7)
  c) Baum, dessen Besetzung in der untersten Ebene zwischen Fall a) und Fall b) liegt

Erwartetes Ergebnis	Verhalten des Programms
Ein Baum ist in der untersten Ebene vollständig gefüllt, wenn die Anzahl k der Punkte $k = 2^n$ beträgt. Also sind als Eingabe Punktmengen mit 4, 8, 16, 32 usw. Punkten zu betrachten. Hier werden 8 Punkte ausgewählt. Es muss sich ein Baum mit 8 Blättern in der dritten Ebene ergeben.	 **Beispiel 5: Ausgeglichener Baum für 8 Punkte**

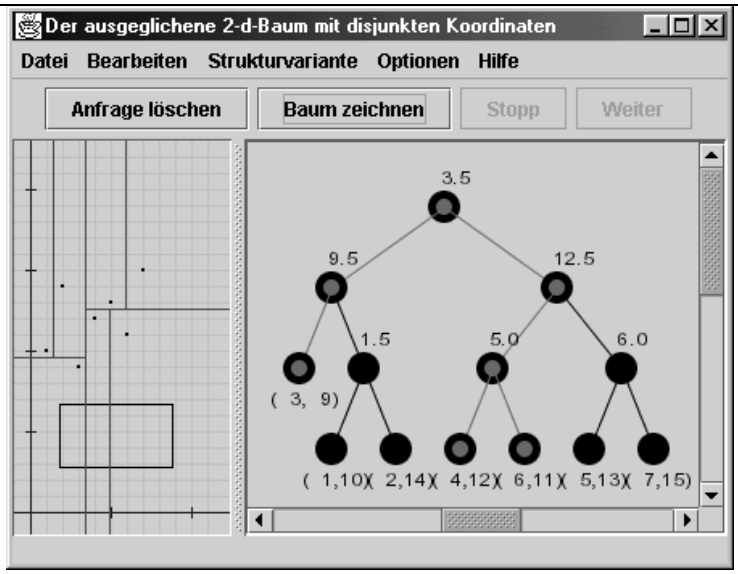
Erwartetes Ergebnis	Verhalten des Programms
Ein ausgeglichener 2-d-Baum hat in der untersten Ebene genau dann 2 Blätter, wenn die Anzahl *k* der Punkte um 1 größer ist als beim Baum von Beispiel 5. Es gilt also: $k = 2^n + 1$. Dabei müssen aufgrund des implementierten Algorithmus diese beiden Blätter am äußersten rechten Knoten angeheftet sein.	 **Beispiel 6: Ausgeglichener Baum für 9 Punkte**

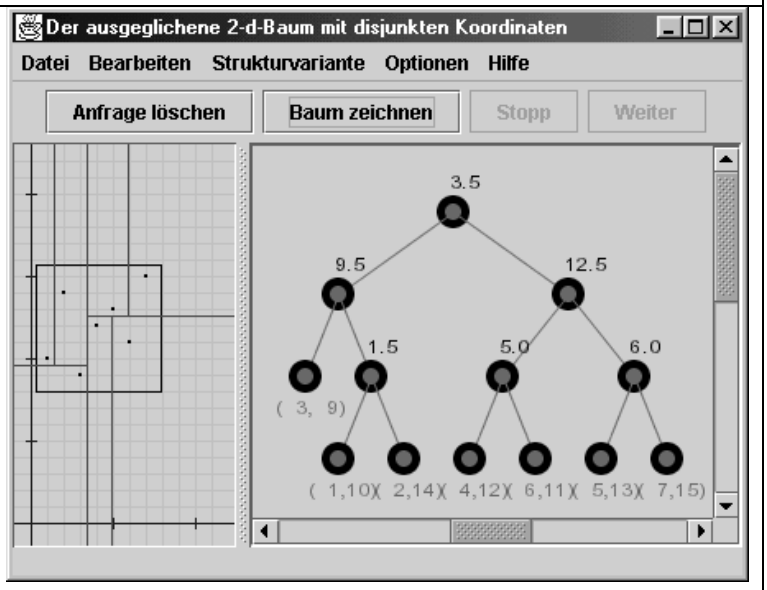
Erwartetes Ergebnis	Verhalten des Programms
Ein ausgeglichener 2-d-Baum hat in der untersten Ebene genau dann 2 Blätter weniger als maximal möglich, wenn die Anzahl *k* der Punkte um 1 kleiner ist als beim Baum von Beispiel 5, also: $k = 2^n - 1$.	 **Beispiel 7: Ausgeglichener Baum für 7 Punkte**

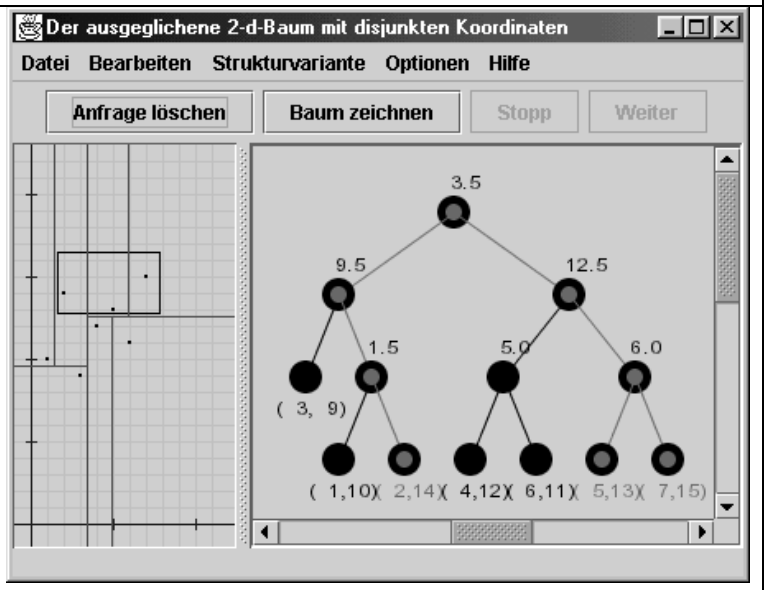
Erwartetes Ergebnis	Verhalten des Programms
Ein ausgeglichener 2-d-Baum hat in der untersten Ebene genau dann eine zwischen Beispiel 6 und Beispiel 7 liegende Besetzung, wenn für die Anzahl *k* der Punkte für gilt: $2^n + 1 < k < 2^n - 1$ mit $n > 2$.	 **Beispiel 7: Ausgeglichener Baum für 11 Punkte**

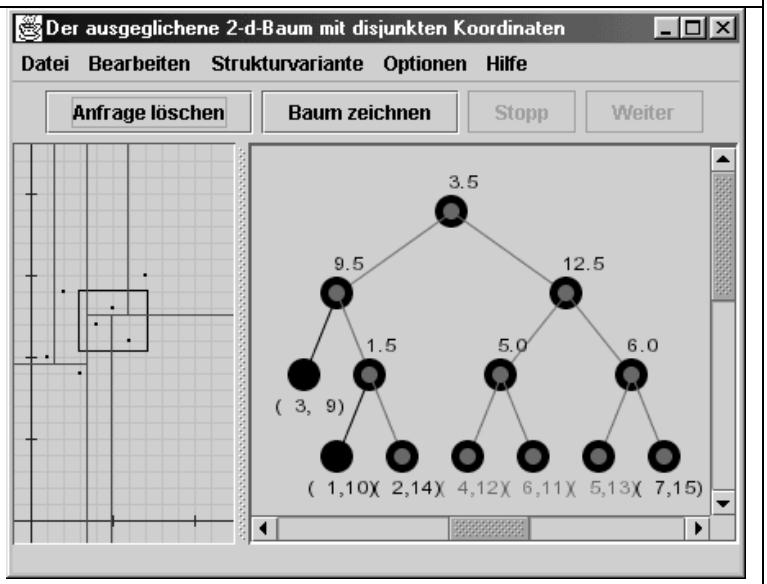
Für die Anfrage im ausgeglichenen 2-d-Baum werden folgende Eingabeäquivalenzklassen betrachtet:

- Es befindet sich kein Punkt im Anfragerechteck (s. Beispiel 8).
  Dieser Fall lässt sich in weitere Unterfälle aufteilen. So kann das Anfragerechteck komplett in einem Splitbereich liegen oder mehrere Splitbereiche schneiden.
- Es liegen alle Punkte im Anfragerechteck (s. Beispiel 9).
- Es liegen einzelne Punkte im Anfragerechteck:
  - Das Anfragerechteck bedeckt ausschließlich die Splitbereiche der enthaltenen Punkte (s. Beispiel 10).
  - Das Anfragerechteck schließt auch andere Splitbereiche ohne deren Punkte ein (s. Beispiel 11).

Erwartetes Ergebnis	Verhalten des Programms
Im Beispiel müssen die drei Blätter als angefasst grün gekennzeichnet werden, deren zugeordnete Splitbereiche vom Anfragerechteck geschnitten werden. Die in den Blättern gespeicherten Punkte müssen schwarz bleiben, weil sie außerhalb des Anfragerechtecks liegen. Alle Pfade zu den besuchten Blättern sollen ebenfalls gekennzeichnet sein.	 **Beispiel 8: Anfrage mit leerem Anfragerechteck**

Erwartetes Ergebnis	Verhalten des Programms
Befinden sich alle Punkte im Anfragerechteck, so müssen alle Knoten besucht werden. Dies soll durch grüne Einfärbung der Knoten erkennbar sein. Da alle Punkte im Anfragerechteck liegen, müssen auch sie grün dargestellt werden.	 **Beispiel 9: Anfrage mit allen Punkten im Anfragerechteck**

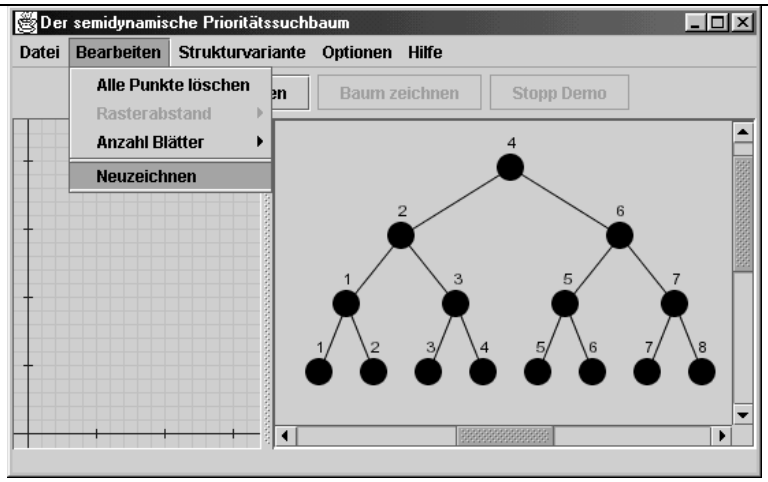
Erwartetes Ergebnis	Verhalten des Programms
Die zu den Punkten im Anfragerechteck gehörenden Blätter sind in grün als angefasst zu kennzeichnen. Des weiteren sollen die Punkte dieser Blätter in grün dargestellt werden. Alle Knoten auf den Wegen zu diesen Blättern müssen ebenfalls als angefasst gekennzeichnet sein.	 **Beispiel 10: Anfragerechteck nur in den Splitbereichen der enthaltenen Punkte**

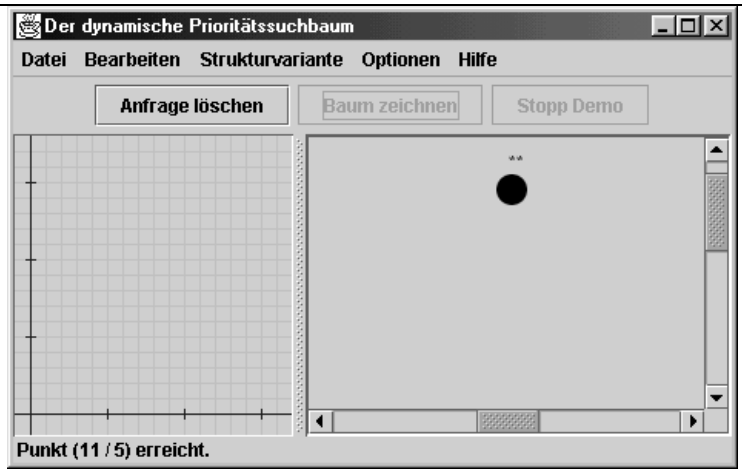
Erwartetes Ergebnis	Verhalten des Programms
Die zu den Punkten im Anfragerechteck gehörenden Blätter sind in grün als angefasst zu kennzeichnen. Auch deren eingetragene Punkte müssen grün dargestellt werden. Alle Pfade zu diesen Blättern müssen ebenfalls markiert sein. Die Blätter zu den Splitbereichen, die geschnitten werden, ohne dass ihre Punkte innerhalb des Anfragerechtecks liegen, werden als besucht gekennzeichnet. Die dort gespeicherten Punkte bleiben jedoch schwarz.	 **Beispiel 11: Anfragerechteck auch in Splitbereichen ohne deren Punkte einzuschließen**

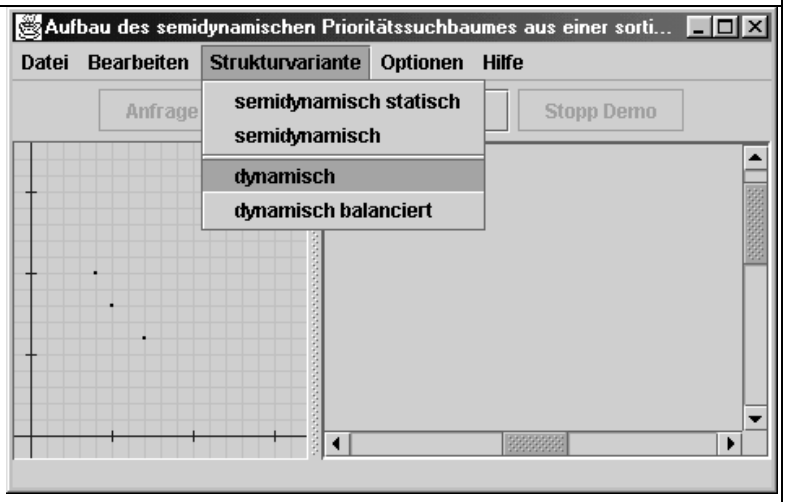
9.2 Tests zum Prioritätssuchbaum

Beispiele für den Test spezieller Werte

Typische Testfälle hierzu ergeben sich beim Überprüfen der leeren Datenstrukturen, beim Laden und Speichern einer leeren Punktliste (siehe hierzu Kapitel 9.1) und beim Wechsel zwischen dem semidynamisch-statischen Prioritätssuchbaum und den anderen Strukturvarianten.

Erwartetes Ergebnis	Verhalten des Programms
Bei der Auswahl der Strukturvariante semidynamisch oder nach dem Löschen aller Punkte muss der leere Skelettbaum aufgebaut werden. Gleiches gilt für die Auswahl des Menüpunktes „Neuzeichnen“ bei leerer Punktliste.	 **Beispiel 12: Test des Aufbaus eines leeren semidynamischen Baums**

Erwartetes Ergebnis	Verhalten des Programms
Bei Auswahl der Strukturvarianten „dynamisch“, „dynamisch balanciert“ oder nach dem Löschen aller Punkte muss der initiale Knoten des dynamischen Prioritätssuchbaumes mit dem speziellen Splitwert ∞ im Ausgabebereich dargestellt werden.	 **Beispiel 13: Test des Aufbaus eines leeren dynamischen Baums**

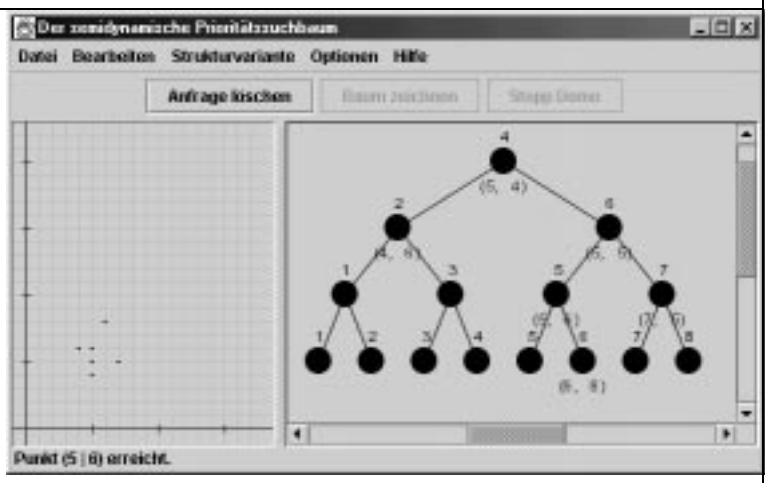
Erwartetes Ergebnis	Verhalten des Programms
Hat der Benutzer im semi-dynamisch-statischen Fall Punkte eingegeben, aber den Baum über die Schaltfläche „Baum zeichnen“ noch nicht aufbauen lassen, so muss der Wechsel in die anderen Strukturvarianten trotzdem ungestört ablaufen, auch wenn beim statischen Fall der Baumcontroller „leer“ ist, solange noch kein Baum existiert.	 **Beispiel 14: Wechsel vom semidynamisch-statischen Baum zu anderen Strukturvarianten**

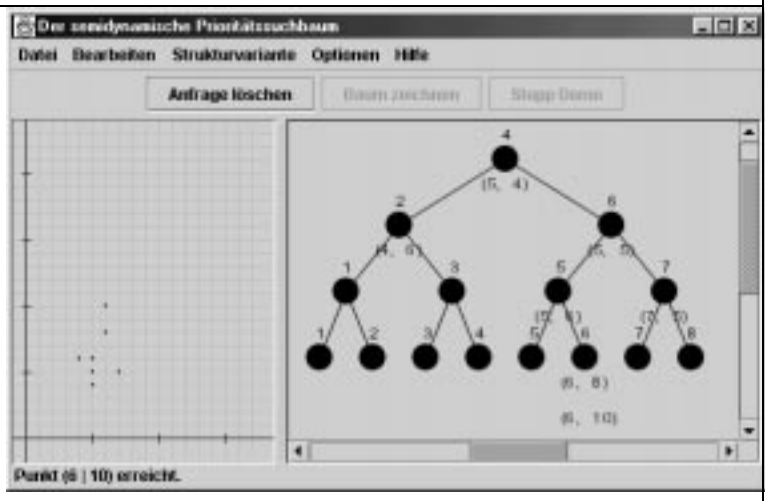
Beispiele für das Testen funktionaler Äquivalenzklassen

Exemplarisch folgen Beispiele zum Testen von Einfügevorgängen, Löschsituationen und Bereichsanfragen bei verschiedenen Strukturvarianten. Wenn möglich, erfolgt dabei eine Einteilung der Eingabe- oder Ausgabedaten in Äquivalenzklassen mit einer Auswahl konkreter Testwerte für die Beispiele.

Beim Einfügen in den semidynamischen Prioritätssuchbaum mit allgemeiner Punktmenge ergeben sich zwei Fälle für das Ausgabeverhalten:

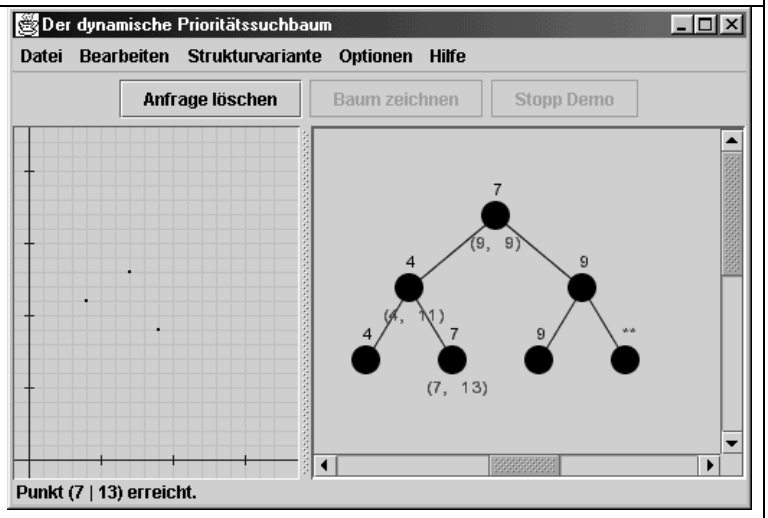
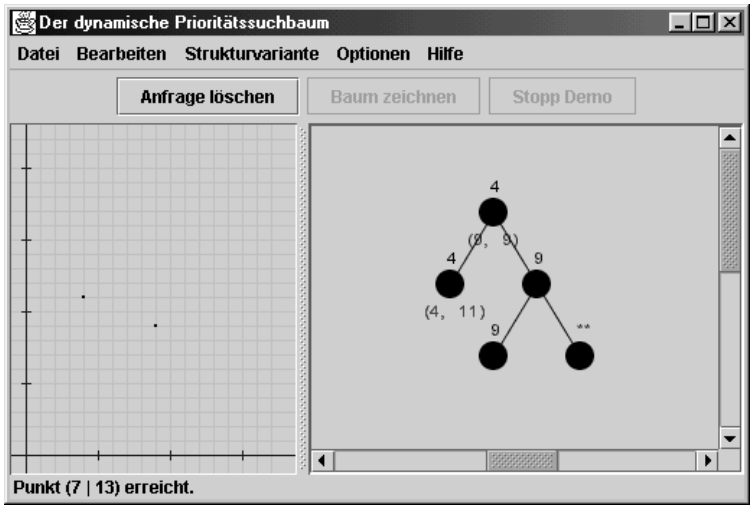
- Der einzufügende Punkt findet im Baum Platz (Beispiel 20).
- Der einzufügende Punkt muss in die Liste des zugehörigen Blattes eingefügt werden, weil der Einfügepfad im Baum bereits voll besetzt ist (Beispiel 21).

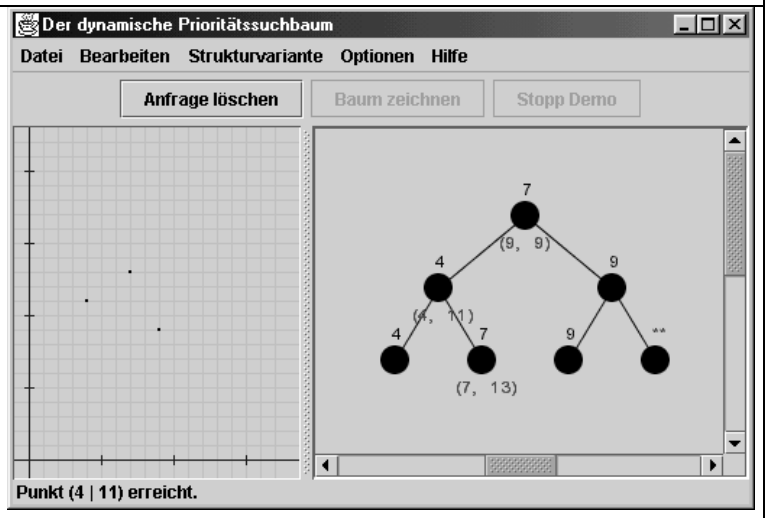
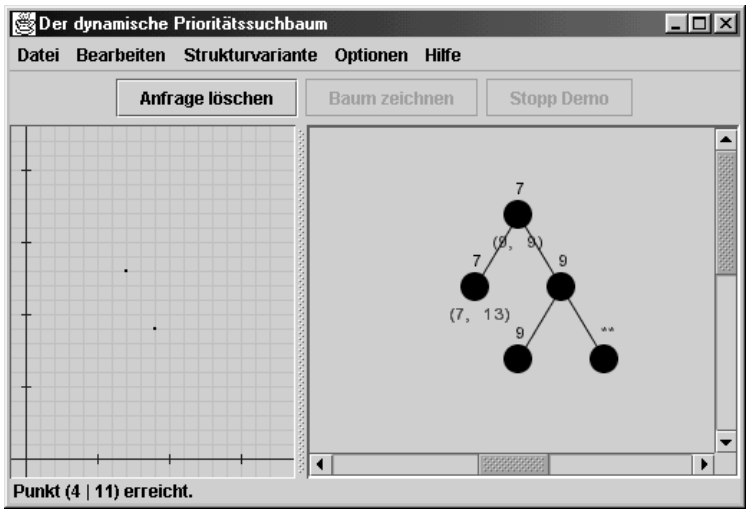
Erwartetes Ergebnis	Verhalten des Programms
Beim Einfügen des Punktes (5|6) muss der Punkt (6|8) aus dem inneren Knoten 5 in den Blattknoten 6 verdrängt werden. Der Punkt (5|6) soll im inneren Knoten 5 des Baums seinen Platz finden.	 **Beispiel 19: Einfügen im semidynamischen Baum bei allgemeiner Punktmenge 1**

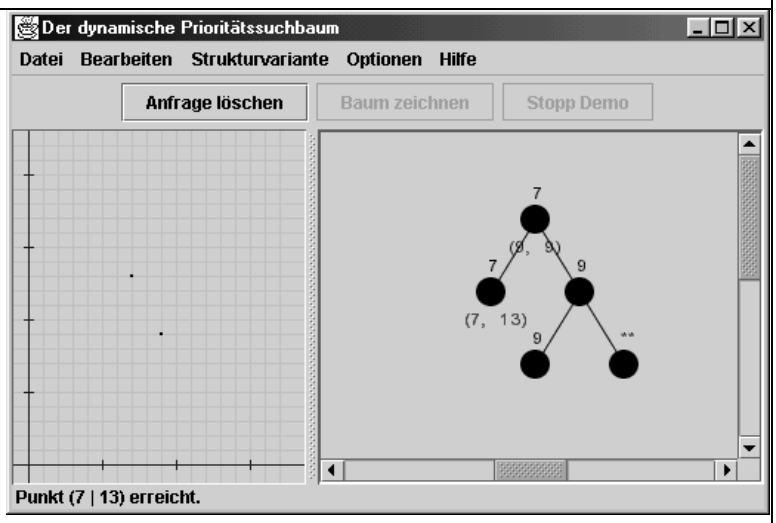
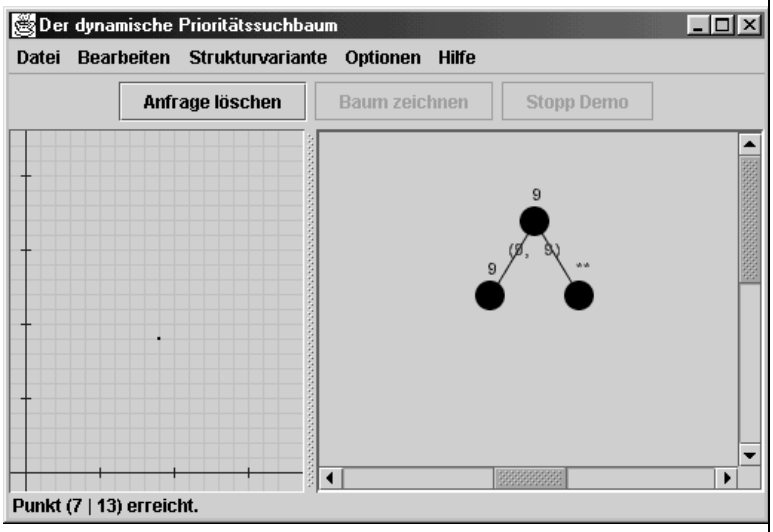
Erwartetes Ergebnis	Verhalten des Programms
Das Einfügen des Punktes (6|10) kann keine vorhandenen Punkte verdrängen. Da der gesamte Einfügepfad zum Blatt 6 bereits belegt ist, muss der Punkt (6|10) im Blatt 6 in die Punktliste eingetragen werden. Da die Punktliste noch leer ist, muss der Punkt (6|10) als erster Eintrag der Liste erscheinen.	 **Beispiel 20: Einfügen im semidynamischen Baum bei allgemeiner Punktmenge 2**

Beim Löschen im dynamischen Prioritätssuchbaum lassen sich folgende Ausgabeäquivalenzklassen betrachten:

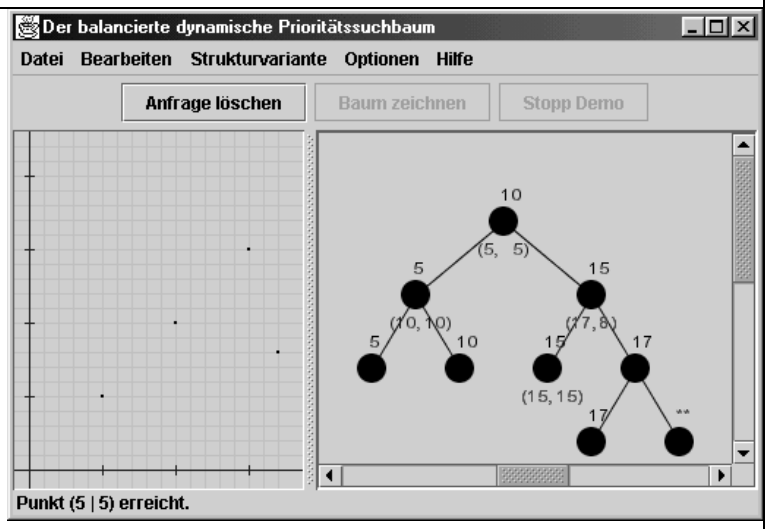
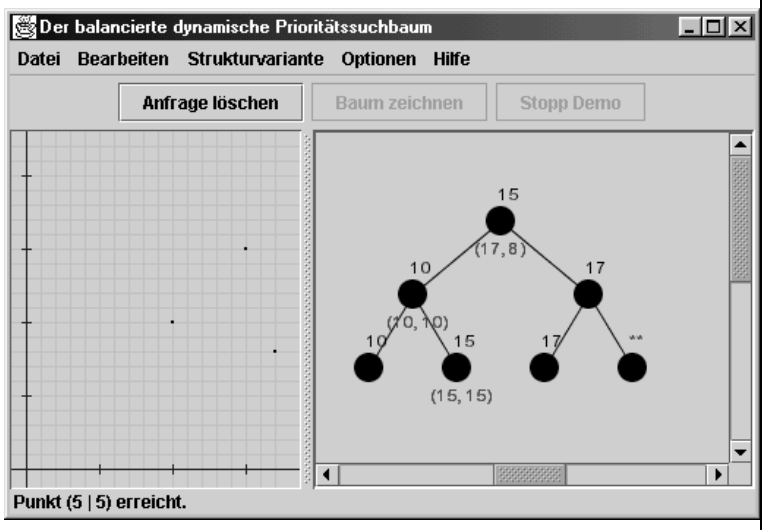
- Die zu löschenden Knoten bilden ein getrenntes Knotenpaar (Beispiel 16)
- Die zu löschenden Knoten bilden ein zusammenhängendes Knotenpaar.
  Dabei können wir weiter unterteilen in den Spezialfall
  - Wurzel und linker Sohn (Beispiel 18) und
  - sonstige Fälle (Beispiel 19)

Erwartetes Ergebnis	Verhalten des Programms
Im Beispiel wird der Punkt (7|13) gelöscht. Der Blattknoten 7 und der innere Knoten 7 bilden ein getrenntes Knotenpaar. Deshalb wird der innere Knoten 7 nicht gelöscht. Nur sein Splitwert ist zu korrigieren, nachdem der Blattknoten 7 und dessen Vorgängerknoten 4 gelöscht wurden. Dabei wird das Blatt 4 linker Nachfolger des inneren Knotens 7, dessen Splitwert dann auf 4 (den Nachfolgerwert) korrigiert wird. Der Punkt (4|11) muss beim Löschvorgang natürlich erhalten bleiben. Er muss vor dem Löschen der Knoten in den verbleibenden Nachfolgerknoten (Blatt 4) verlagert werden.	  **Beispiel 16: Löschen im dynamischen Baum 1**

Erwartetes Ergebnis	Verhalten des Programms
Im Beispiel wird der Punkt (4|11) gelöscht. Der Blattknoten 4 und der innere Knoten 4 bilden ein zusammenhängendes Knotenpaar. Beim Löschen muss das Blatt 7 linker Nachfolger des inneren Knotens mit dem Splitwert 7 werden. Da nach dem Löschen des Punktes (4|11) der Punkt (7|13) in den inneren Knoten 4 hochgezogen wird, muss sicher gestellt sein, dass der Punkt trotz des Löschens erhalten bleibt. Er muss vor dem Entfernen der Knoten wieder in den verbleibenden Nachfolgerknoten (Blatt 7) verschoben werden.	  **Beispiel 17: Löschen im dynamischen Baum 2**

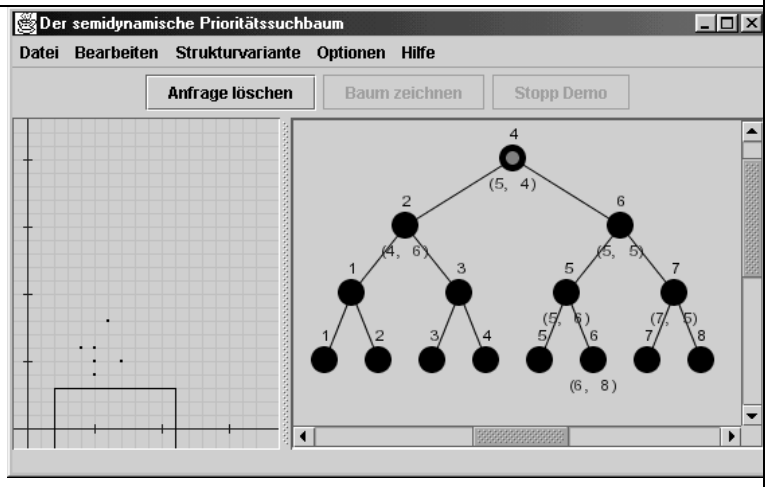
Erwartetes Ergebnis	Verhalten des Programms
Im Beispiel wird der Punkt (7|13) gelöscht. Der Blattknoten 7 und der innere Knoten 7 bilden ein zusammenhängendes Knotenpaar. Als Spezialfall weist der innere Knoten als Baumwurzel aber keinen Vorgänger auf. Deshalb muss der rechte Nachfolgerknoten der Wurzel zum neuen Wurzelknoten werden. Der Punkt (9|9) muss vor dem Löschen des Wurzelknotens in den rechten Nachfolgerknoten 9 verschoben werden.	  **Beispiel 18: Löschen im dynamischen Baum 3**

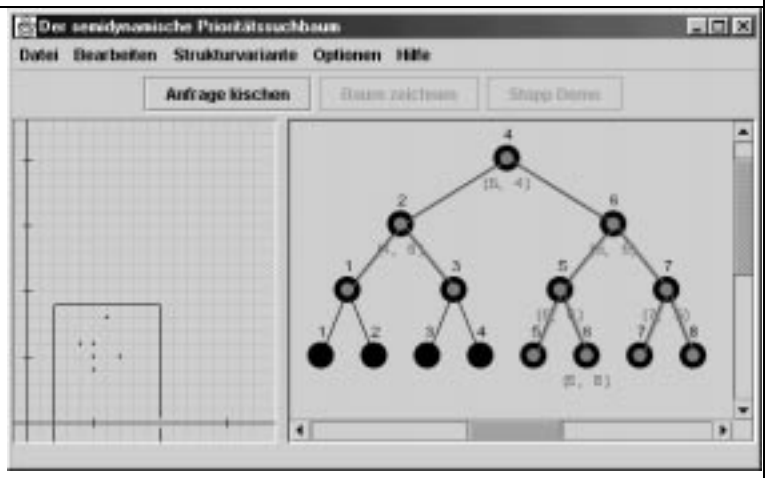
Nachdem wir bereits verschiedene Löschvorgänge betrachtet und dabei darauf hingewiesen haben, dass öfter Punkte vor dem Löschen von Knoten an eine neue Position verschoben werden müssen, soll ein letzter spezieller Löschvorgang getestet werden. Im dynamisch balancierten Baum muss besonders darauf geachtet werden, dass beim Löschen die Ordnung im y-Heap nicht verletzt wird (Beispiel 15).

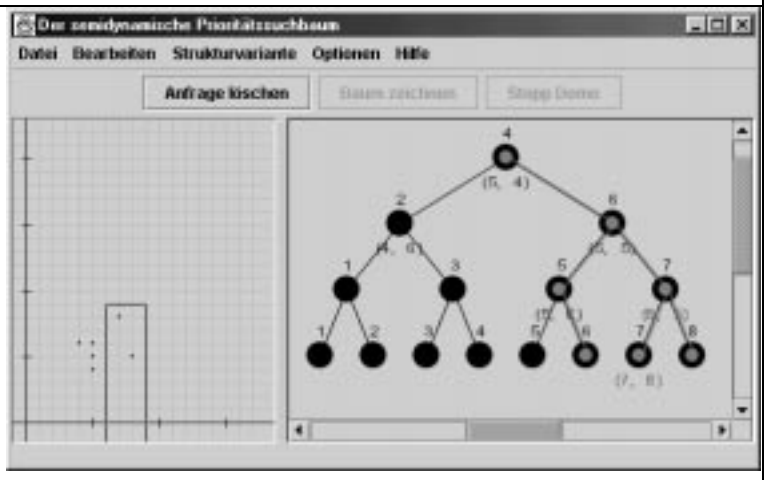
Erwartetes Ergebnis	**Verhalten des Programms**
Der Punkt (5|5) wird gelöscht. Das zusammenhängende Knotenpaar mit dem Blattknoten 5 und dem inneren Knoten 5 wird wie im dynamischen Fall üblich gelöscht. Dadurch verlagern sich der Punkt (17|8) in den Wurzelknoten und der Punkt (15|15) in den inneren Knoten 15. Im Wurzelknoten ist das AVL-Kriterium verletzt. Der Baum muss balanciert werden. Bei der erforderlichen Einfachrotation nach links muss die Ordnung im y-Heap erhalten bleiben. Vor dem Rotieren ist der Punkt (15|15) ins Blatt 15 zu verschieben, um die neue Wurzel freizumachen. Nach dem Rotieren mit dem Umhängen des Blattes 15 müssen die Punkte (17|8) und (10|10) nach oben verschoben werden. Die entstandenen Lücken werden geschlossen.	  **Beispiel 15: Löschen im dynamisch balancierten Baum**

Für die Tests zu den Bereichsanfragen im semidynamischen Prioritätssuchbaum wird eine Aufteilung in folgende Eingabeäquivalenzklassen vorgenommen:

- Es befindet sich kein Punkt im Anfragehalbstreifen (s. Beispiel 21).
- Es liegen alle Punkte im Anfragehalbstreifen (s. Beispiel 22).
- Es liegen einzelne Punkte im Anfragehalbstreifen (Beispiel 23).

Erwartetes Ergebnis	**Verhalten des Programms**
Der Anfragehalbstreifen beinhaltet keine Punkte. Im Baum muss das aufgezogene x-Intervall von 2 bis 11 durchsucht werden. Die Suche ist abzubrechen, wenn dabei Punkte angetroffen werden, deren y-Koordinate größer als 3 (y-Grenzwert) ist oder ein Knoten keinen Punkt enthält. Die Bereichsanfrage im Beispiel muss bereits im Wurzelknoten abbrechen.	 **Beispiel 19: Anfrage ohne Punkt**

Erwartetes Ergebnis	**Verhalten des Programms**
Alle Punkte liegen im Halbstreifen. Besucht werden muss das markierte x-Intervall von 2 bis 10. Die Suche kann nur noch durch nichtbesetzte Knoten früher abgebrochen werden. Alle Punkte müssen nach Ende der Suche markiert sein.	 **Beispiel 20: Anfrage mit allen Punkten**

Erwartetes Ergebnis	Verhalten des Programms
Durchsucht werden muss das x-Intervall von 6 bis 9. Dabei sind die Punkte (7|8) und (8|5) als gefunden zu markieren. Die Suche soll bei Punkten mit y > 9 oder in leeren Knoten abbrechen. Im Beispiel müssen deshalb alle drei Pfade bis zu den Blättern durchlaufen werden.	 **Beispiel 21: Anfrage mit ausgewählten Punkten**

10. Bedienungsanleitung des Programms VisualGeom

10.1 Systemvoraussetzungen

Das Programm **`VisualGeom`** wurde von uns auf folgenden Computern entwickelt und getestet:

- SUN UltraSparc 144MHz mit 64MB unter Solaris 2.6
- Pentium 266MMX mit 128MB unter Linux mit der SuSE-Distribution in der Version 6.1
- Pentium II 233MHz mit 128MB unter Windows95
- Pentium 266MMX mit 128MB unter Windows98

Auf allen Systemen konnte auch bei der Animation ein ansprechender Bewegungsablauf erzielt werden. Die Geschwindigkeit beim Test mit einem Pentium 166 mit 64MB unter Windows95 war gerade noch vertretbar. Aufgrund dieser Ergebnisse erscheint uns unter Windows und Linux als Minimalanforderung ein Pentium-Rechner mit mindestens 200MHz angeraten. Dabei werden an freiem Speicherplatz auf der Festplatte für die Installation von JDK2 mindestens 90MByte, für JRE2 mindestens 25MByte und für **`JavaHelp`** mindestens 12MByte benötigt.
Für das Nutzen des Programms als Applikation gelten die Rahmenbedingungen von Tabelle 3.

Betriebssystem	**Erforderliche Software**
Windows 95 Windows 98 Windows NT	• JDK2 oder JRE2 • JavaHelp oder die Datei **`jhall.jar`** aus dem JavaHelp-System • abhängig von der jeweiligen JDK2-Version ggf. das compilierte Paket **`com.sun.image.codec.jpeg`**
Linux	• JDK2 oder JRE2 • JavaHelp oder die Datei **`jhall.jar`** aus dem JavaHelp-System • das compilierte Paket **`com.sun.image.codec.jpeg`**
Solaris	• JDK2 oder JRE2 • JavaHelp oder die Datei **`jhall.jar`** aus dem JavaHelp-System • das compilierte Paket **`com.sun.image.codec.jpeg`**

Tabelle 3: Für die Applikation benötigte Software

Für die Nutzung des Programms als Applet gelten die Rahmenbedingungen von Tabelle 4.

Betriebssystem	**Erforderliche Software**
Windows 95 Windows 98 Windows NT	Bei Einsatz des Netscape Communicators ab Version 4.x oder des Internet Explorers ab Version 4.x: • JRE2 einschließlich Plug-In • aus JavaHelp die Datei **`jhall.jar`** • abhängig von der jeweiligen JDK2-Version ggf. das compilierte Paket **`com.sun.image.codec.jpeg`** Bei Einsatz des **`appletviewer`** der Firma SUN: • das Programm **`appletviewer`** aus JDK2 • JRE2 • aus JavaHelp die Datei **`jhall.jar`** • abhängig von der jeweiligen JDK2-Version ggf. das compilierte Paket **`com.sun.image.codec.jpeg`**
Linux	• JRE2 • aus JavaHelp die Datei **`jhall.jar`** • das compilierte Paket **`com.sun.image.codec.jpeg`** **Hinweis**: Das Plug-In für den Netscape Communicator befindet sich zur Zeit noch in der Entwicklung. Der aktuelle Stand ist unter **`http://www.blackdown.org/`** erfahrbar. Deshalb ist die Nutzung als Applet ausschließlich mit dem im JDK2 enthaltenen Programm **`appletviewer`** möglich.
Solaris	• JRE2 • Plug-In für JRE2 • aus JavaHelp die Datei **`jhall.jar`** • das compilierte Paket **`com.sun.image.codec.jpeg`** • Netscape in der Version 4.x oder höher

Tabelle 4: Für das Applet benötigte Software

10.2 Installation

Die CD aus Kapitel 4 des Anhangs enthält neben **VisualGeom** für Solaris, Windows und Linux auch JDK2 und JRE2 einschließlich der aktuell verfügbaren Plug-Ins für diese drei Betriebssysteme. Des weiteren sind folgende Dateien vorhanden:

- **jhall.jar**
  Sie enthält die für **VisualGeom** benötigten Klassen des Java-Hilfesystems.
- **codec.jar**
  Diese Datei enthält die für VisualGeom benötigten compilierten Klassen des zu JDK2 gehörenden Paketes **com.sun.image.codec.jpeg.**

Nach der Installation von JDK2 bzw. JRE2 sind **jhall.jar** und **codec.jar** in das Verzeichnis von JDK bzw. das unter JRE befindliche lib-Verzeichnis zu kopieren. Anschließend müssen sie in den Klassenpfad mit Hilfe der Umgebungsvariablen **CLASSPATH** aufgenommen werden.

- Mit JDK2 unter Windows im Verzeichnis **C:\JDK2** lautet der Eintrag in der Datei **AUTOEXEC.BAT**:
  ```
  SET CLASSPATH=C:\JDK2\src.jar;C:\JDK2\codec.jar;.;C:\JDK2\jhall.jar
  ```
- Unter Linux ergibt sich mit JDK2, **jhall.jar** und **codec.jar** im Verzeichnis **/usr/lib/java/** z. B. folgender Eintrag in der Datei **/etc/profile:**
  ```
  CLASSPATH=/usr/lib/java/src.jar: .:/usr/lib/java/lib/dt.jar
  CLASSPATH=$CLASSPATH:/usr/lib/java/lib/tools.jar
  CLASSPATH=$CLASSPATH:/usr/lib/java/jhall.jar
  CLASSPATH=$CLASSPATH:/usr/lib/java/codec.jar
  export CLASSPATH
  ```
  Alternativ kann dieser Eintrag in der benutzerspezifischen Einstellungsdatei **.profile** im Benutzerheimatverzeichnis vorgenommen werden.
- Für JDK2 unter Solaris sind abhängig vom Verzeichnis, in das JDK2 installiert wurde, entsprechende Einstellung wie unter Linux durchzuführen.

Die **VisualGeom**-Programmdateien sind mit der vorhandenen Verzeichnisstruktur in ein beliebiges Verzeichnis auf der Festplatte zu entpacken. Dabei wird zunächst das Unterverzeichnis **VisualGeom** angelegt. In dieses werden dann alle Verzeichnisse und Dateien für die Benutzung als Applikation und Applet kopiert.

- Windows95, 98, NT

  Entpacken Sie mit **WinZip** oder einem anderem Programm, das ZIP-Dateien mit langen Dateinamen entpacken kann, die Datei **VisualGeom.zip** in das von Ihnen ausgewählte Verzeichnis.

  Alternativ besteht die Möglichkeit, das Programm **VisualGeom.exe** aufzurufen. Es entpackt als ausführbares ZIP-Archiv selbständig die Programmdateien in das von Ihnen ausgewählte Verzeichnis.

- Linux und Solaris

  Mounten Sie die CD z. B. in das Verzeichnis **/cdrom**. Kopieren Sie die Datei **VisualGeom.tgz** in das Verzeichnis, in das Sie **VisualGeom** installieren wollen, z. B. **/home/kirk**. Entpacken Sie mit **tar** die Datei **VisualGeom.tgz** in das von Ihnen ausgewählte Verzeichnis:

  ```
  cp –R /cdrom/VisualGeom/ VisualGeom VisualGeom.tgr
  tar –xvf VisualGeom.tgz
  ```

  Beachten Sie bitte, dass Sie ggf. die Rechte und Benutzeridentifikationen anpassen müssen.

10.3 Einschränkungen

Einschränkungen ergeben sich aktuell für die Applikation bei folgenden Punkten:

- Die Zeit für die Durchführung der Druckmethode ist bedingt durch die spezifische Art der Implementation der Firma SUN unvertretbar langsam.

- Der zeichenbare Bereich wurde von uns wegen der Größe des Ausgabefensters auf maximal 10 Ebenen und maximal 32 Baumknoten in der Ebene 5 beschränkt. Bei „ungeeigneter" Lage

der Knoten können in den tiefer liegenden Ebenen abhängig von der Art des schöner Zeichnens auch weniger Knoten sichtbar sein. Mit einer Fehlermeldung wird darauf hingewiesen, wenn ein Knoten nicht mehr im sichtbaren Bereich liegt. Die Eingabe ist allerdings nicht gesperrt.

- Das Speichern von JPEG-Dateien unter Solaris führt zu einer Fehlermeldung.

Bei Anwendung von VisualGeom als Applet gelten wegen des Betriebs in der Sandbox folgende weitere Einschränkungen:

- Das Speichern und Laden von Punktdateien ist nicht möglich.
- Das Speichern von Eingabe- und Ausgabebereich als JPEG-Datei ist nicht möglich.
- Das Drucken aus einem Applet heraus ist nicht möglich.
- Die Nutzung des Java-Hilfesystems bereitet beim Aufruf des Applets mit dem **`appletviewer`** Schwierigkeiten.

10.4 Starten der Anwendung

Für das Starten als Applikation und als Applet ist in das Verzeichnis **`VisualGeom`** zu wechseln. In diesem befinden sich die Startdatei **`VisualGeom.class`** für den Aufruf als Applikation, **`VisualGeom.html`** für den Start als Applet mit dem Programm **`appletviewer`** und **`VisualGeomPlugIn.html`** für den Start als Applet mit dem Netscape Communicator oder dem Internet Explorer, wenn JRE2 einschließlich Plug-In installiert ist.
Dabei ist **`VisualGeomPlugIn.html`** eine mit dem **`HTMLConverter`** von SUN modifizierte HTML-Datei. Diese wird für die Nutzung mit den Browsern von Netscape und Microsoft benötigt.

- Start als Applikation aus einem Textfenster

  `java VisualGeom`

- Start als Applet

Browser	Aufruf
Netscape Communicator	Den Netscape Communicator starten und dann die Datei **`VisualGeomPlugIn.html`** laden.
Internet Explorer	Den Internet Explorer starten und dann die Datei **`VisualGeomPlugIn.html`** laden.
appletviewer	Das Programm wird aus einem MSDOS- oder xterm-Fenster aufgerufen mittels: **`appletviewer VisualGeom.html.`**

Tabelle 5: Aufruf von `VisualGeom` als Applet

Zusätzlich lassen sich auch die Programme für den 2-d-Baum oder den Prioritätssuchbaum einzeln als Applikation oder Applet im Verzeichnis **`VisualGeom`** aufrufen.

- Start als Applikation aus einem Textfenster

  `java kd/KdApplication` für den 2-d-Baum

  `java prio/PrioApplication` für den Prioritätssuchbaum

- Start als Applet

  `appletviewer KdApplet.html` für den 2-d-Baum

  `appletviewer PrioApplet.html` für den Prioritätssuchbaum

  Alternativ kann **`KdAppletPlugIn.html`** oder **`PrioAppletPlugIn.html`** mit dem Netscape Communicator oder dem Internet Explorer geladen werden.

10.5 Das Startmenü

Das Startmenü stellt drei Buttons zur Verfügung. Angeboten werden der 2-d-Baum, der Prioritätssuchbaum und das Beenden der Applikation bzw. das Schließen des Applets.

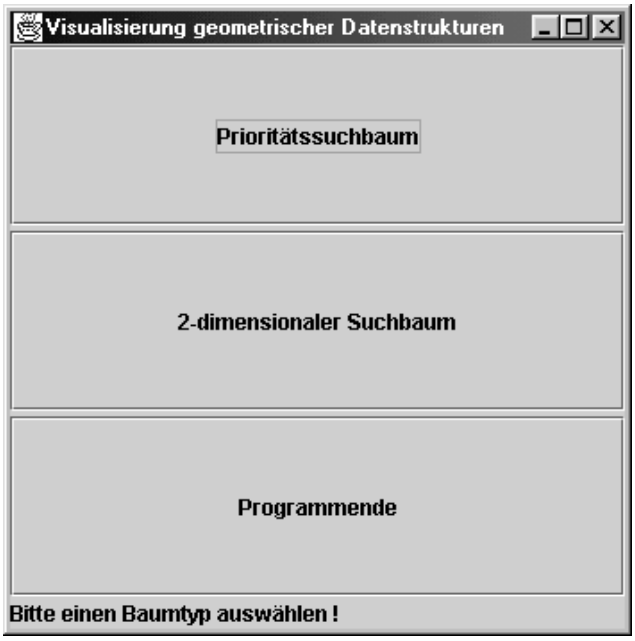


Abbildung 78: Startmenü der Applikation von `VisualGeom`

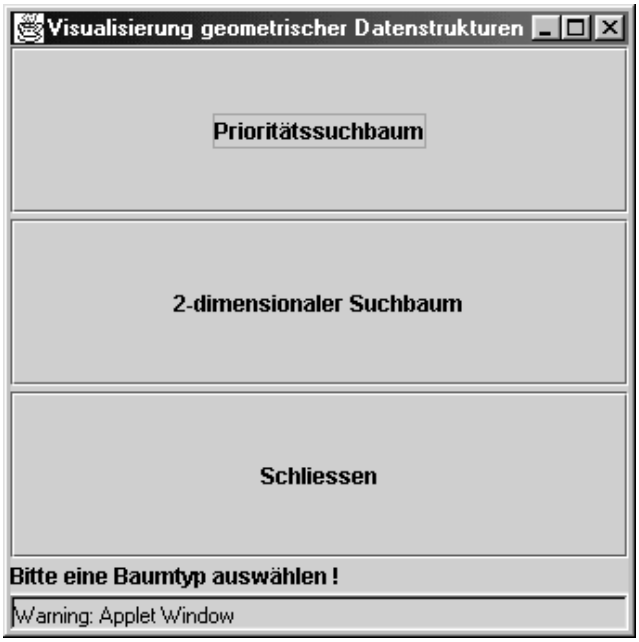


Abbildung 79: Startmenü des Applets von `VisualGeom`

10.6 Der 2-d-Baum

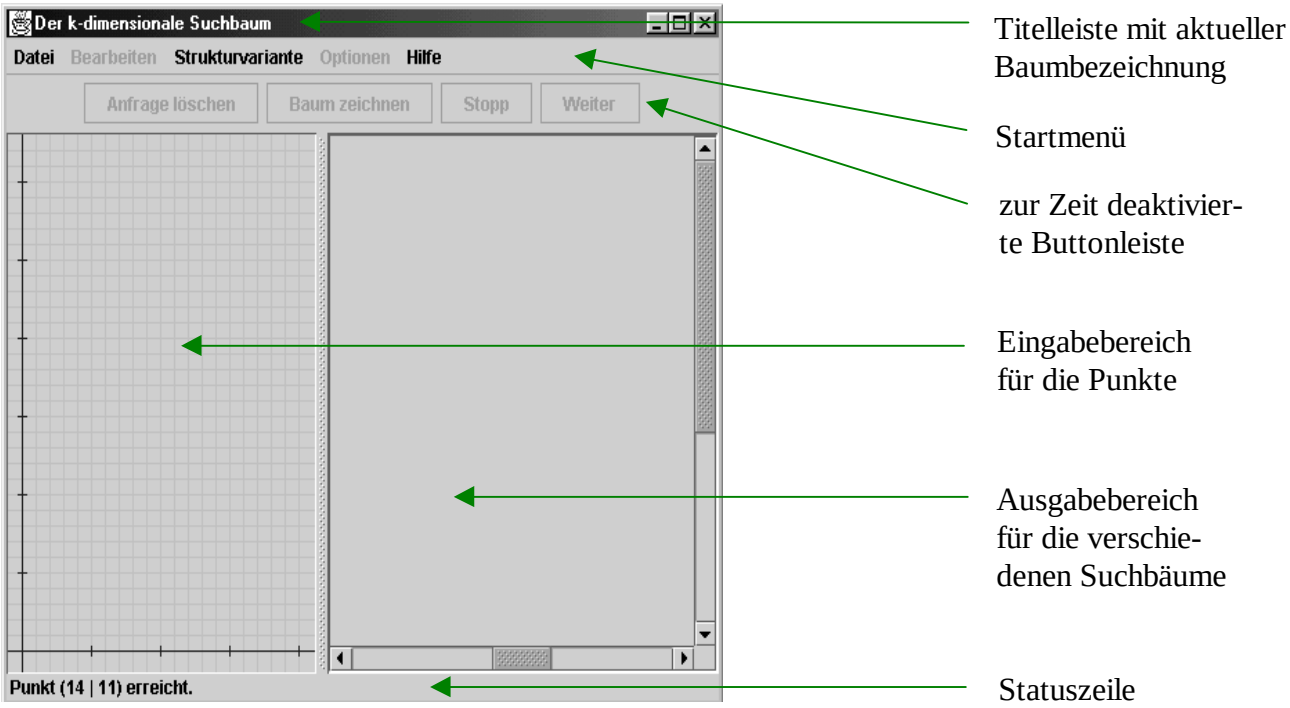


Abbildung 80: Startfenster des 2-d-Baums

Erst nach der Auswahl einer Strukturvarianten können mit der Maus Punkte im Eingabebereich eingegeben oder gelöscht und Anfragerechtecke aufgezogen werden. Dabei gelten folgende Zuordnungen:

- Linke Maustaste im Eingabebereich

  Das Klicken mit der linken Maustaste bewirkt, dass ein Punkt eingegeben wird. Ist dieser Punkt bereits vorhanden oder handelt es sich bei den 2-d-Bäumen um solche, bei denen sich die Punkte in allen Koordinaten voneinander unterscheiden müssen[84], erfolgt die Ausgabe einer Fehlermeldung in der Statuszeile.

[84] Die gilt für den dynamischen, den dynamisch interaktiven und den ausgeglichenen 2-d-Baum. Nur der allgemeine 2-d-Baum kann Punkte mit identischer Koordinate verwalten.

- Linke Maustaste im Ausgabebereich

  Wird im Ausgabebereich ein allgemeiner 2-d-Baum angezeigt, der innere Knoten mit einer zusätzlichen blauen Markierung enthält, so deutet dies auf vorhandene 1-d-Bäume hin. Diese können durch Anklicken der betreffenden Knoten in eigenen Fenstern betrachtet werden.

- Rechte Maustaste im Eingabebereich

  Das Anklicken eines Punktes mit der rechten Maustaste bewirkt das Löschen des Punktes im Eingabebereich und im Baum.

- Ziehen der Maus mit gedrückter linker Maustaste im Eingabebereich

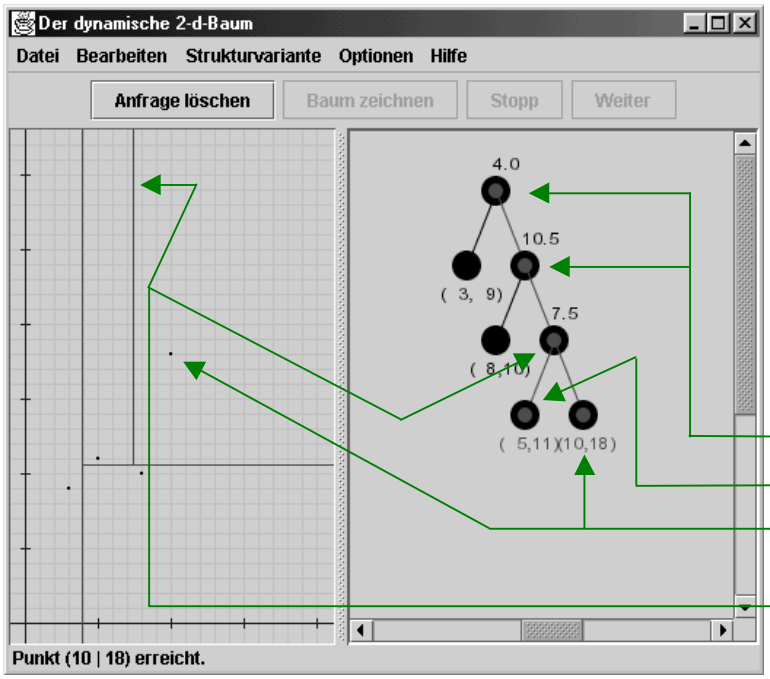
  Dies führt zum Platzieren und Öffnen eines Anfragerechtecks.

Die Buttons in der Buttonleiste sind abhängig vom jeweiligen Programmmodus aktiviert oder deaktiviert. Ihre Bedeutung kann der Tabelle 6 entnommen werden.

Button	**Bedeutung**
Anfrage löschen	Mit dem Betätigen dieses Buttons werden die Anfragemarkierungen im 2-d-Baum und das Anfragerechteck in der Eingabefläche gelöscht.
Baum zeichnen	Erst nach dem Betätigen dieses Buttons erfolgt bei den statischen Varianten ausgewogener und allgemeiner 2-d-Baum der Aufbau des Suchbaums.
Stopp	Hiermit kann der animierte Aufbau eines 2-d-Baums oder die Demonstration zum dynamischen bzw. ausgewogenen 2-d-Baum beendet werden.
Weiter	Beim schrittweisen Aufbau des ausgewogenen oder allgemeinen 2-d-Baums wird die nächste Ebene dargestellt.

Tabelle 6: Bedeutung der vier Buttons beim 2-d-Baum

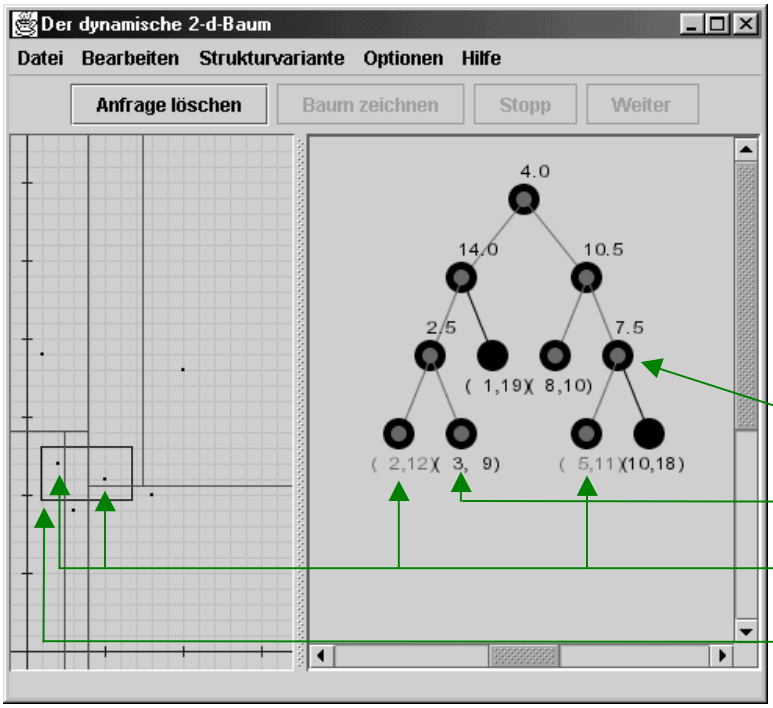
10.6.1 Interpretation der Ausgaben zum 2-d-Baum



Beim dynamischen 2-d-Baum werden im animierten Modus alle Knoten rot gekennzeichnet, die angefasst werden müssen, um den Punkt (10,18) einzufügen. Das Blatt, das diesen Punkt enthält, ist ebenfalls rot gekennzeichnet.

angefasste Knoten
umgehängter Knoten
neu eingefügter Punkt und Knoten
neu eingefügter Splitknoten und neue Splitgerade

Abbildung 81: Animierter dynamischer 2-d-Baum



Wird ein Anfragerechteck eingezeichnet, werden alle bei der Anfrage angefassten Knoten durch grüne Kreise gekennzeichnet. Liegt ein Punkt im Anfragerechteck, wird zusätzlich der Punktwert grün dargestellt.

angefasster innerer Knoten
angefasstes Blatt
Punkte in Anfragerechteck
Anfragerechteck

Abbildung 82: Anfrage durchführen

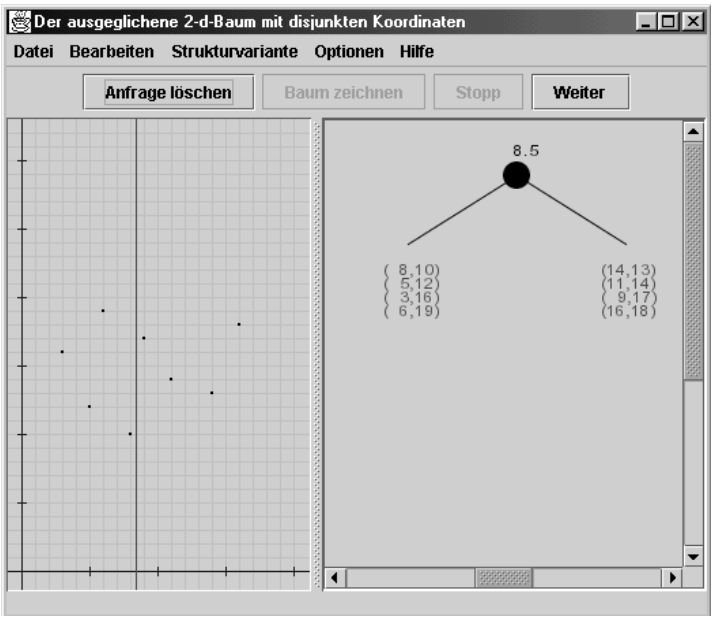


Abbildung 83: Schrittweiser Aufbau des ausgeglichenen 2-d-Baums

Zunächst werden die Punkte eingegeben. Das Betätigen des Buttons „Baum zeichnen“ bewirkt die Durchführung des ersten Schrittes. Die Punktemenge wird zunächst nach der x-Koordinate sortiert. Dann wird als arithmetisches Mittel der benachbarten x-Koordinaten 8 und 9 der Splitwert 8,5 berechnet. Die Punktmenge wird in zwei gleichmächtige oder sich um maximal 1 unterscheidende Teilmengen aufgeteilt. Diese werden nach der y-Koordinate sortiert und sind in rot dargestellt.

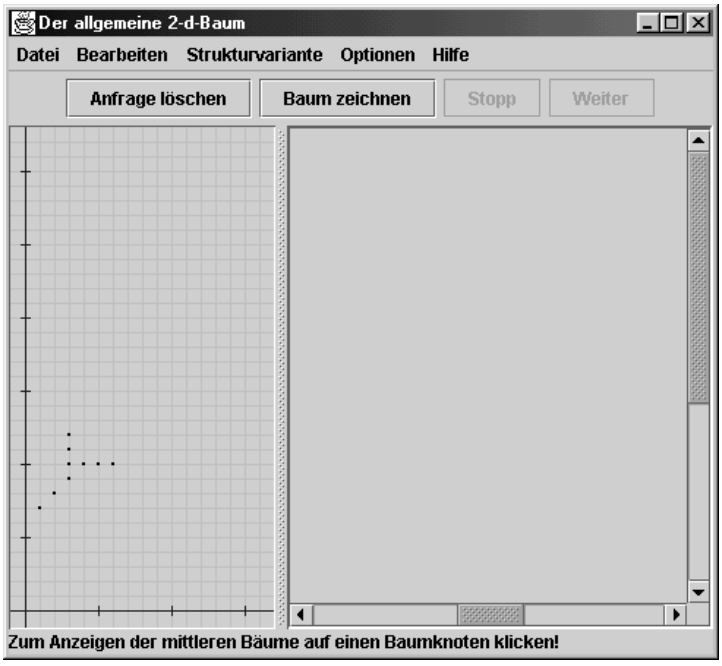
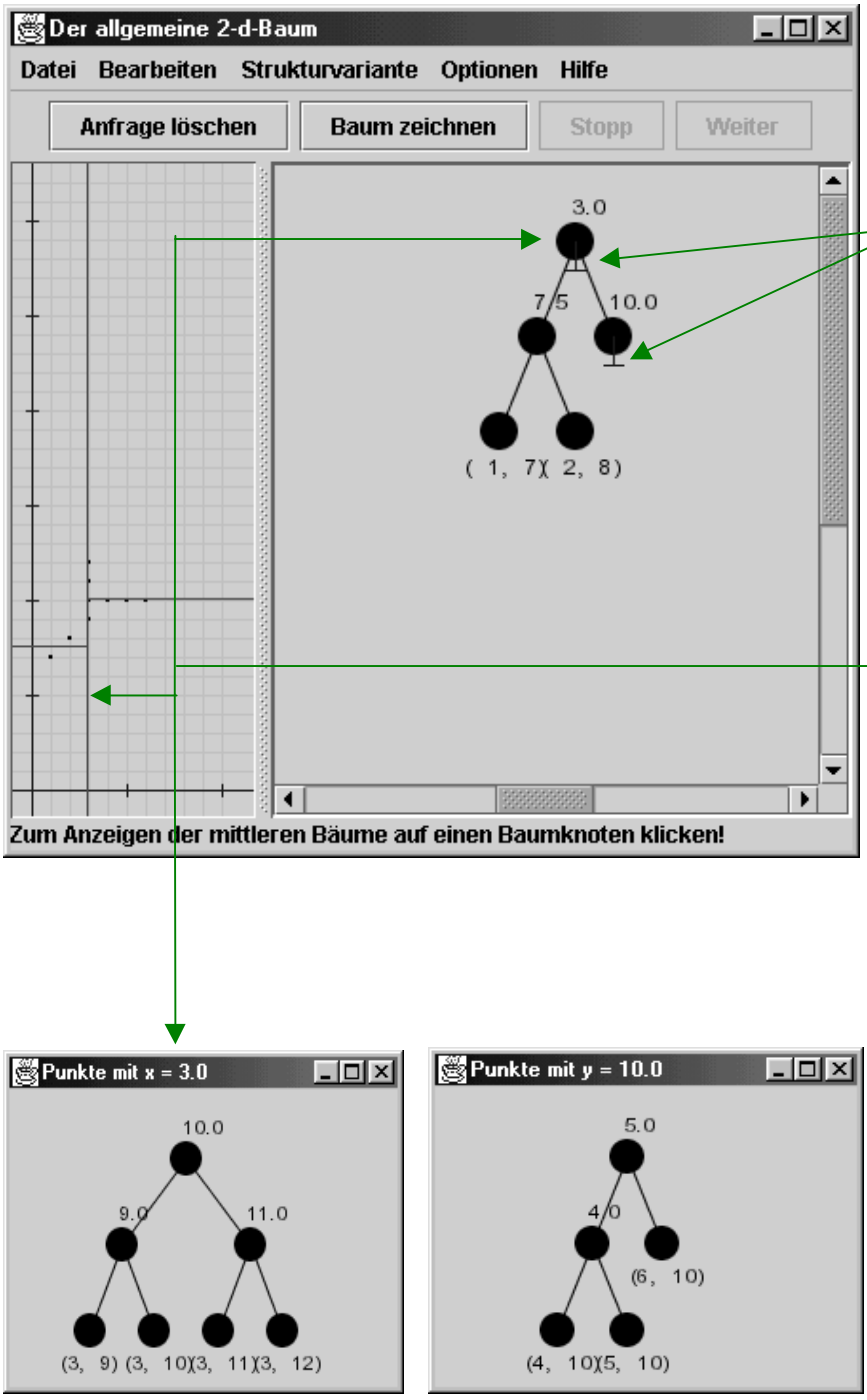


Abbildung 84: Allgemeiner 2-d-Baum (Teil 1)

Aus dem Eingabebereich von Abbildung 84 wird ersichtlich, dass vier Punkte mit identischer x-Koordinate und drei weitere mit identischer y-Koordinate vorkommen. Bei Aufteilung der nach der x-Koordinate sortierten Punktmenge ergibt sich x=3 als Splitwert (Abbildung 85). Alle Punkte mit dieser x-Koordinate werden in einem 1-d-Baum gespeichert, der am Knoten mit dem Splitwert x=3 angehängt ist. Klicken mit der linken Maustaste auf diesen Knoten bewirkt das Öffnen eines Fensters mit dem zugeordneten 1-d-Baum (Abbildung 85).



Hierdurch wird gekennzeichnet, dass am Knoten ein 1-d-Baum existiert.

Splitgerade für x=3 mit 4 Punkten gleicher x-Koordinate, zugeordnetem Splitknoten mit Hinweis auf einen 1-d-Baum und separatem Fenster für die Darstellung dieses 1-d-Baums

Abbildung 85: Allgemeiner 2-d-Baum

10.6.2 Das Menu Datei zum 2-d-Baum

Neu	Das Programm wird in den Startzustand zurückgesetzt. Ggf. geöffnete Dateien werden geschlossen, Dateinamen gelöscht und die ausgewählte Strukturvariante zurückgesetzt. Die Menüpunkte unter Optionen werden ebenfalls auf die Startwerte zurückgesetzt. In der Titelleiste erscheint die Meldung „**Der 2-dimensionale Suchbaum**“. Alle in diesem Modus nicht nutzbaren Menüpunkte und alle Buttons werden deaktiviert.
Öffnen	Beim ersten Mal wird das Dateiverzeichnis **`VisualGeom`** geöffnet, sonst das aktuell eingestellte. Die ausgewählte Strukturvariante bestimmt den Filter für die Dateianzeige. • **`*.kd`** Dateien mit Punkten, die sich in allen Koordinaten voneinander unterscheiden • **`*.kda`** Dateien mit allgemeiner Punktmenge
Speichern	Die abgespeicherte Punktmenge wird aktualisiert.
Speichern unter	Der Benutzer hat die Möglichkeit, die Punktliste als Textdatei auf der Festplatte in einem auswählbaren Verzeichnis zu sichern. Bei erstmaligem Speichern wird hier das Verzeichnis **`Visualgeom`** geöffnet, sonst das aktuell eingestellte Verzeichnis. Die Dateiendung **`kd`** bzw. **`kda`** wird automatisch als Suffix an den Dateinamen angehängt.
Drucken	Der Inhalt von Ein- und Ausgabefenster wird auf einem Drucker ausgegeben.
Schließen	Die Bearbeitung des Suchbaums wird beendet und die Fenster geschlossen. Wurde der 2-d-Baum aus dem Startmenü aufgerufen, so wird dieses angezeigt. Andernfalls wird die Applikation oder das Applet beendet.
Beenden	Das Arbeiten mit dem Programm **`VisualGeom`** wird beendet.

Tabelle 7: 2-d-Menüpunkte des Datei-Menüs

10.6.3 Das Menü Bearbeiten zum 2-d-Baum

Alle Punkte löschen	Alle Punkte werden aus der Punktliste entfernt. Die Punkte aus dem Eingabebereich und der 2-d-Baum aus dem Ausgabebereich gelöscht. Der Benutzer hat damit die Möglichkeit, die Punkte erneut einzugeben. Wurde bereits die Punktliste in einer Datei gespeichert, bleibt dieser Dateiname erhalten. Gleiches gilt auch für die Einstellung unter „Optionen“ und „Art des Zeichnens“.
Rasterabstand	Der Abstand zwischen den Rasterlinien zum „Einfangen“ der Punkte ist einstellbar. Zur Auswahl stehen 10, 5 und 2 Bildpunkte Abstand zwischen den ganzzahligen x- bzw. y-Koordinaten. Liegt der Abstand bei 2, so werden keine Rasterlinien angezeigt.
Neuzeichnen	Die Inhalte des Eingabe- und des Ausgabefensters werden gelöscht. Abhängig von der ausgewählten Strukturvariante und der Einstellung unter „Optionen – Art des Zeichnens“ erfolgt anhand der Punktliste der Neuaufbau von Punkten, Splitgeraden und 2-d-Baum. Dabei wird jeweils nur die aktuelle Punktliste genutzt. Gelöschte Punkte haben keinen Einfluss auf den Aufbau des jeweiligen 2-d-Baums.

Tabelle 8: 2-d-Menüpunkte des Bearbeiten-Menüs

10.6.4 Das Menü Strukturvariante zum 2-d-Baum

War bereits eine der Strukturvarianten aus Tabelle 9 oder Tabelle 10 ausgewählt und wird diese erneut oder eine andere gewählt, so werden Punktliste, Punkte und Suchbaum gelöscht.
Welche Strukturvariante gerade ausgewählt ist, kann der Titelleiste entnommen werden.

dynamischer 2-d-Baum	Nach der Eingabe eines Punktes erfolgt unmittelbar der Aufbau des aktualisierten Suchbaums im Ausgabebereich und Einzeichnen der Splitgeraden im Eingabebereich. Der Splitwert wird dafür als arithmetisches Mittel aus den Koordinaten der beiden im gleichen Splitrechteck befindlichen Punkte berechnet. Das Löschen eines Punktes führt zum Entfernen des zugeordneten Blattes aus dem Baum und der Splitgeraden aus dem Eingabebereich. Bei Durchführen einer Anfrage werden alle angefassten Knoten markiert. Die ausgewählten Blattknoten werden gesondert gekennzeichnet.
dynamisch interaktiver 2-d-Baum	Ist bereits ein Punkt eingefügt, so wird nach der Eingabe jedes weiteren Punktes zunächst gefragt, nach welcher Koordinate (x oder y) der Splitbereich geteilt werden soll. Anschließend muss ein konkreter Splitwert eingegeben werden. Nach erfolgreicher Kontrolle werden die Ergänzungen vorgenommen. Das Löschen eines Punktes führt zum Entfernen des zugeordneten Blattes aus dem Baum und der Splitgeraden aus dem Eingabebereich. Bei Durchführen einer Anfrage werden alle angefassten Knoten markiert. Die ausgewählten Blattknoten werden gesondert gekennzeichnet. Die Menüpunkte Öffnen und Speichern sind deaktiviert, weil in dieser Variante die Splitwerte nicht als arithmetisches Mittel ausgerechnet, sondern vom Benutzer bestimmt werden.
ausgeglichener 2-d-Baum	Zunächst ist die Eingabe und das Löschen von Punkten möglich. Der Aufbau eines ausgeglichenen 2-d-Baums erfolgt erst, wenn der Button „Baum zeichnen" angeklickt worden ist. Die Darstellung des Baumaufbaus ist abhängig von der Einstellung unter „Optionen – Art des Zeichnens". Bei Durchführen einer Anfrage werden alle angefassten Knoten markiert. Die ausgewählten Blattknoten werden gesondert gekennzeichnet.

Tabelle 9: 2-d-Menüpunkte des Menüs Strukturvariante (Teil 1)

allgemeiner 2-d-Baum	Zunächst ist die Eingabe und das Löschen von beliebigen Punkten möglich. Der Aufbau eines allgemeinen 2-d-Baums erfolgt erst, wenn der Button „Baum zeichnen" angeklickt worden ist. Die Darstellung des Baumaufbaus ist abhängig von der Einstellung unter „Optionen – Art des Zeichnens". Ist an einem inneren Knoten eine zusätzliche blaue Markierung angebracht, so deutet dies auf einen angehängten 1-d-Baum hin. Wird ein so markierter Knoten mit der linken Maustaste angeklickt, öffnet sich ein kleines Fenster, das den am Knoten angehängten 1-d-Baum darstellt. Bei Durchführen einer Anfrage werden alle angefassten Knoten markiert. Die ausgewählten Blattknoten werden gesondert gekennzeichnet.

Tabelle 10: 2-d-Menüpunkte des Menüs Strukturvariante (Teil 2)

10.6.5 Das Menü Optionen zum 2-d-Baum

Grafik in Datei	Ein Abbild vom „Eingabebereich" oder „Ausgabebereich" kann als Datei im JPEG-Format auf der Festplatte abgespeichert werden, um es später mit einem Textverarbeitungsprogramm oder mit einem Grafikprogramm weiter zu bearbeiten. Dateiname und Verzeichnis sind frei wählbar. Das Suffix **`jpg`** wird automatisch gesetzt.
Art des Zeichnens	Zur Auswahl stehen „schrittweise", „animiert" und „sofortiger Aufbau". Welche dieser Optionen verfügbar ist, hängt von der jeweils ausgewählten Strukturvariante ab.

Tabelle 11: 2-d-Menüpunkte des Menüs Optionen

10.6.6 Das Menü Hilfe zum 2-d-Baum

Hilfethemen	Angeboten wird eine Online-Anleitung zur Benutzung von **`VisualGeom`**. Über deren Inhalts- und Indexverzeichnis kann gezielt auf eine bestimmte Hilfedatei zugegriffen werden. Die Modifikation der einzelnen Hilfedateien ist problemlos möglich, da es sich um HTML-Dateien handelt.
Demos	Es kann zwischen zwei Demos zum dynamischen und ausgeglichenen 2-d-Baum gewählt werden. • dynamischer 2-d-Baum Es werden zunächst mehrere Punkte eingefügt. Dabei erfolgt parallel der Aufbau des dynamischen 2-dimensionalen Suchbaums, die Ermittlung des Splitwertes und die Darstellung der Splitgeraden. Dann wird ein Punkt gelöscht und die hierdurch erforderliche Änderung im Suchbaum und bei den Splitgeraden vorgenommen. Abschließend wird ein Anfragerechteck gezeichnet. Alle bei der Bereichsanfrage angefassten Knoten und alle ausgewählten Blattknoten werden markiert. • ausgeglichener 2-d-Baum Zunächst erfolgt nur die Eingabe aller Punkte. Dann wird schrittweise der Suchbaum durch Teilung der sortierten Punktlisten ebenenweise aufgebaut. Dabei werden die jeweils ermittelten Splitgeraden eingezeichnet. Abschließend wird ein Anfragerechteck gezeichnet. Alle bei der Bereichsanfrage angefassten Knoten und alle ausgewählten Blattknoten werden markiert.

Tabelle 12: 2-d-Menüpunkte des Hilfe-Menüs

10.7 Der Prioritätssuchbaum

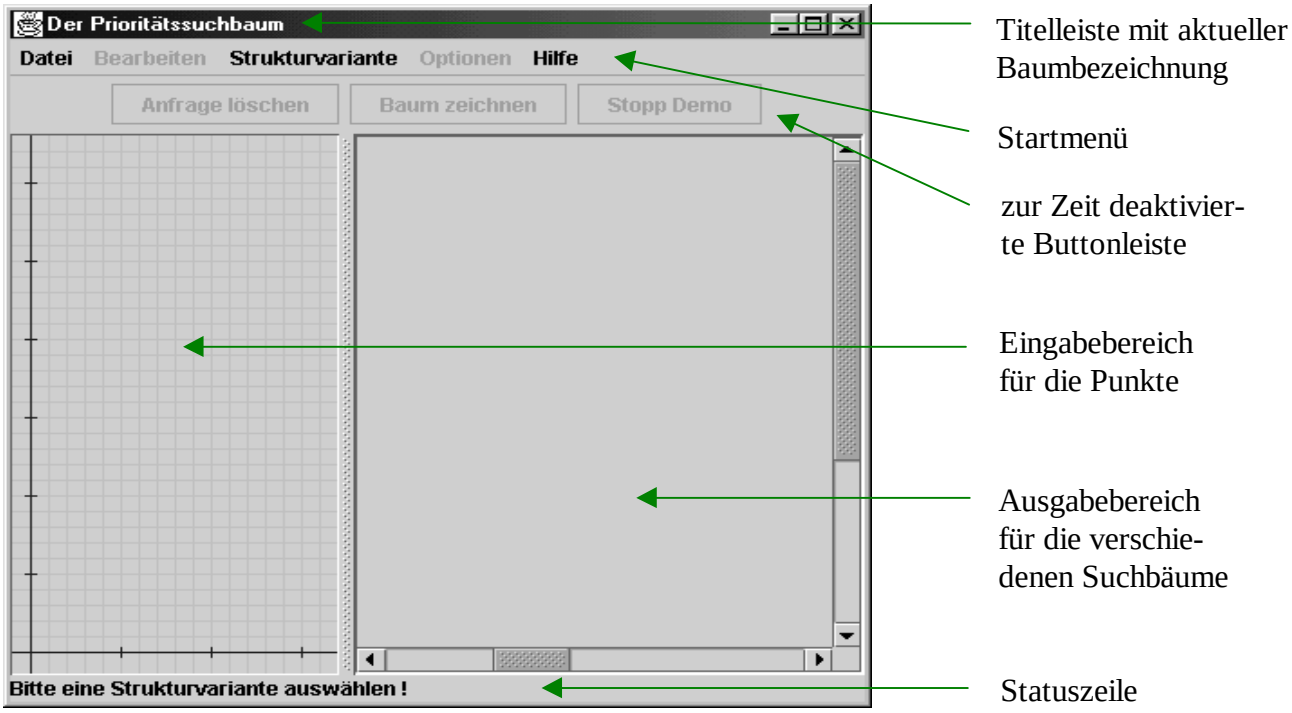


Abbildung 86: Startfenster des Prioritätssuchbaums

Erst nach der Auswahl einer Strukturvarianten können mit der Maus Punkte im Eingabebereich eingegeben oder gelöscht und Anfragehalbstreifen geöffnet werden. Dabei gelten folgende Zuordnungen:

- Linke Maustaste im Eingabebereich

  Das Klicken mit der linken Maustaste im Eingabebereich bewirkt, dass ein Punkt eingegeben wird. Ist dieser Punkt oder seine x-Koordinate bereits vorhanden und eine disjunkte Punktmenge eingestellt, erfolgt die Ausgabe einer Fehlermeldung in der Statuszeile.

- Rechte Maustaste im Eingabebereich

  Das Anklicken mit der rechten Maustaste bewirkt das Entfernen dieses Punktes aus dem Eingabebereich.

- Ziehen der Maus mit gedrückter linker Maustaste im Eingabebereich

  Dies führt zum Platzieren und Öffnen eines Halbstreifens.

Die Buttons in der Buttonleiste sind abhängig vom jeweiligen Programmmodus aktiviert oder deaktiviert. Ihre Bedeutung kann der Tabelle 13 entnommen werden.

Button	Bedeutung
Anfrage löschen	Die Anfragemarkierung im Baum und der Anfragehalbstreifen in der Eingabefläche werden gelöscht..
Baum zeichnen	Erst nach dem Betätigen dieses Buttons erfolgt beim semidynamisch statischen Prioritätssuchbaums der Aufbau des Suchbaums. Wird während eines Baumaufbaus erneut dieser Button betätigt, so beginnt der Baumaufbau von neuem.
Stopp Demo	Die Demos zum Prioritätssuchbaum werden nach Abschluss der aktuellen Aktion beendet.

Tabelle 13: Bedeutung der drei Buttons beim Prioritätssuchbaum

10.7.1 Interpretation der Ausgaben zum Prioritätssuchbaum

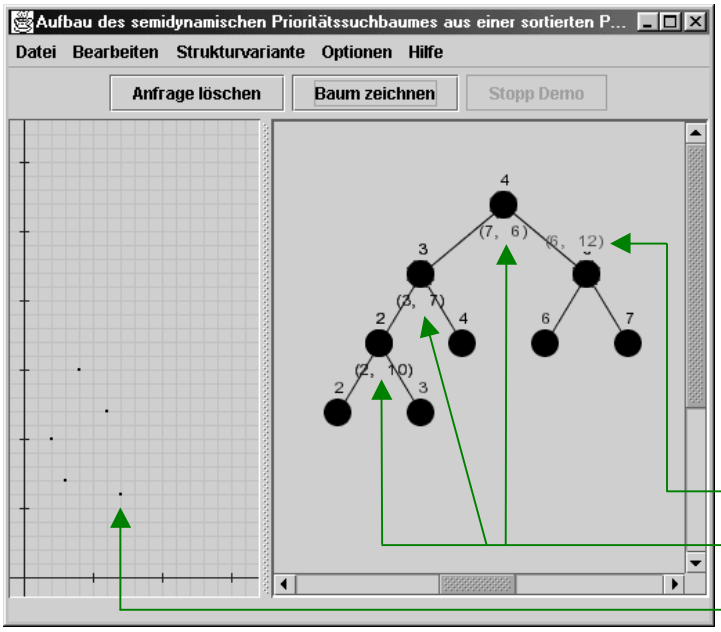


Abbildung 87: Animierter Aufbau beim semidynamisch statischen Prioritätssuchbaum

Zunächst erfolgt die Eingabe aller Punkte. Nach Betätigen des Buttons „Baum zeichnen“ wird der linksvollständige Skelettbaum im Ausgabebereich dargestellt. Die nach der y-Koordinate aufsteigend sortierten Punkte werden nacheinander in den Baum eingefügt.

Einfügen des Punktes (6,12) in den Prioritätssuchbaum

bereits im Baum eingefügte Punkte

noch nicht eingefügter Punkt

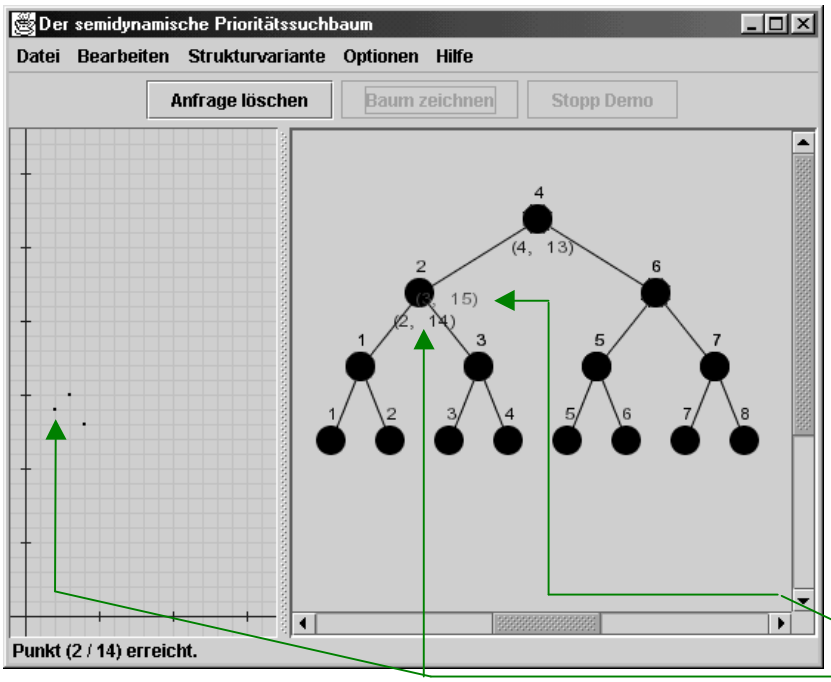


Abbildung 88: Animiertes Punkt-Einfügen in den semidynamischen Prioritätssuchbaum

In den ständig vorhandenen Skelettbaum wird dynamisch jeder neu angeklickte Punkt eingefügt. Dabei lässt sich der Einfügepfad des neuen Punktes bis zu seinem Speicherknoten verfolgen. Verdrängt der neue Punkt einen bereits eingefügten Punkt, so wird auch der Verdrängungsvorgang animiert dargestellt.

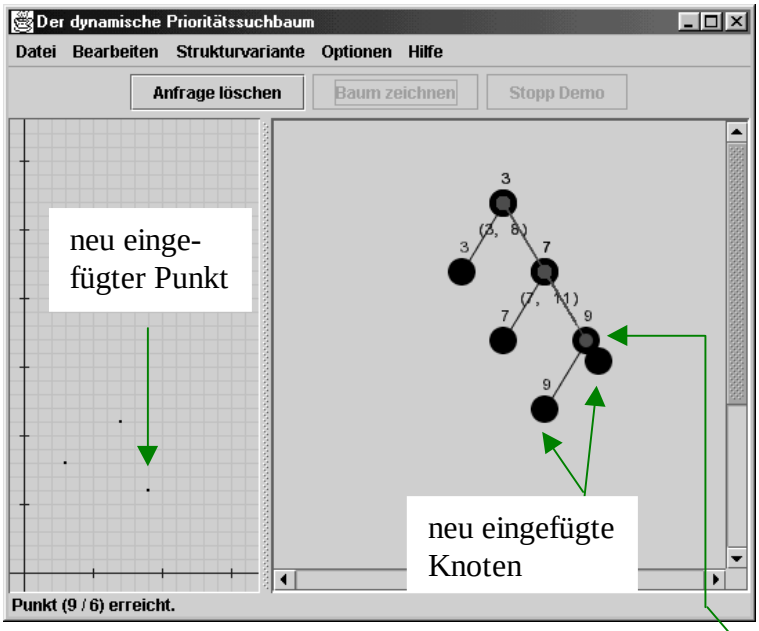
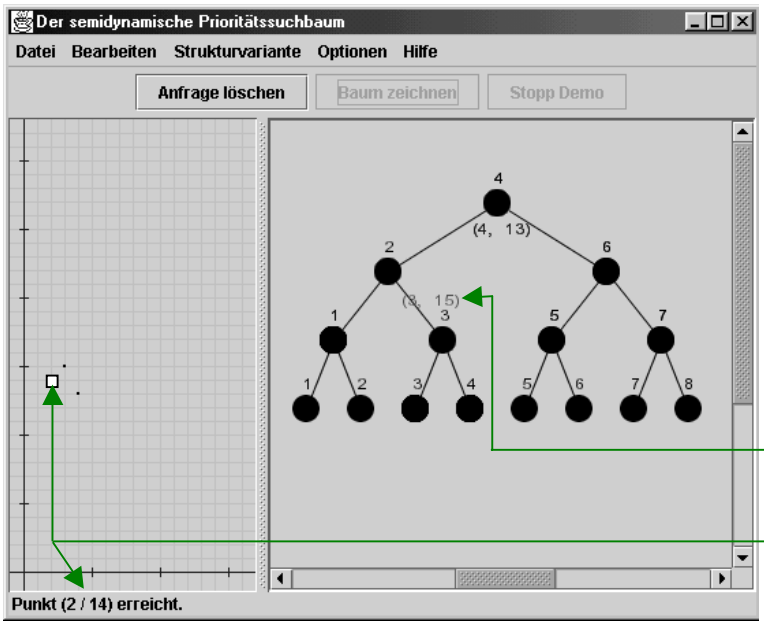


Abbildung 89: Animierter Aufbau beim dynamischen Prioritätssuchbaum

Beim Einfügen eines Punktes in den dynamischen Prioritätssuchbaum wird zunächst die Erweiterung des Skelettbaums visualisiert. Dabei wird der Suchpfad zum Ansatzknoten markiert. Erst wird der neue linke und dann der neue rechte Nachfolgerknoten erzeugt. Ggf. wird der Splitwert im Ansatzknoten korrigiert. Nach der Erweiterung des Skelettbaums wird der neue Punkt wie im semidynamischen Fall eingefügt.



Wird ein Punkt aus dem Eingabebereich entfernt, so wird zunächst der Suchpfad zum Punkt rot eingezeichnet und der Punkt gelöscht. Die entstandene Lücke wird durch „Hochziehen“ von Nachfolgepunkten geschlossen.

nachfolgender Punkte wandert eine Ebene höher

gelöschter Punkt

Abbildung 90: Animiertes Löschen eines Punktes

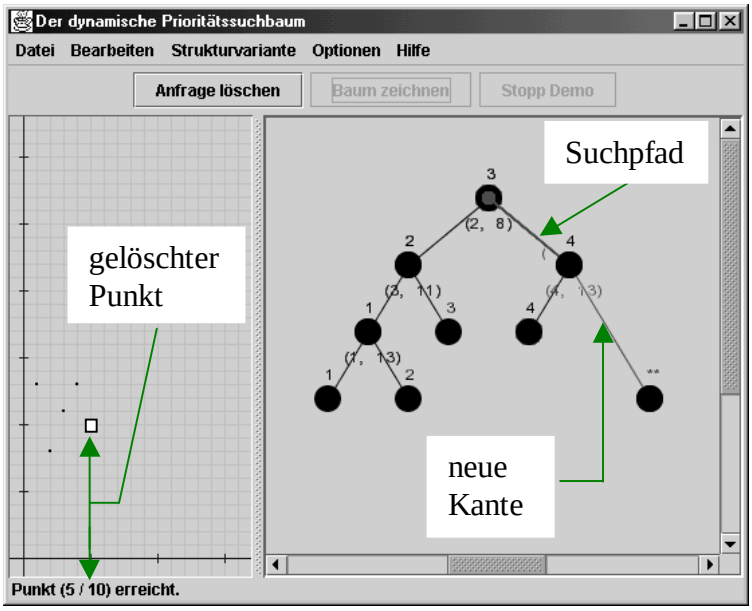
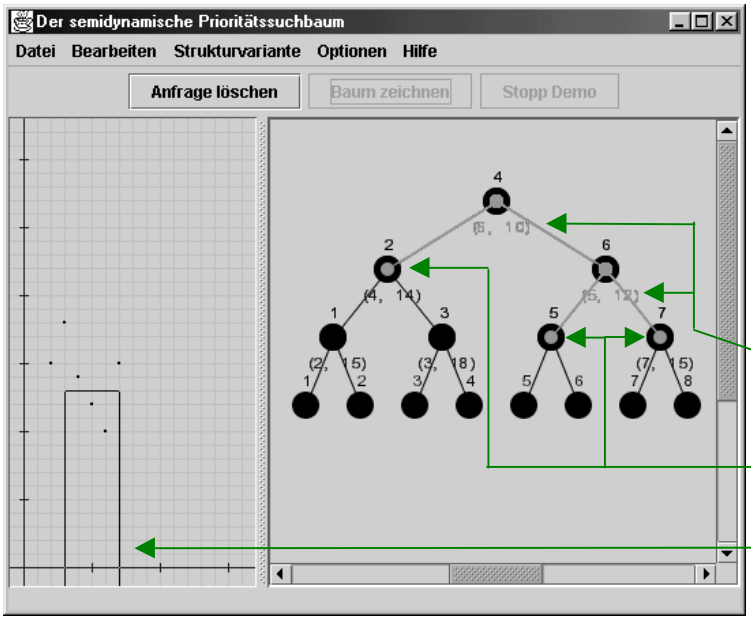


Abbildung 91: Löschen im dynamischen Prioritätssuchbaum

Nach dem Löschen des Punktes wie im semidynamische Fall muss im dynamischen Baum zusätzlich das Löschen der zugehörigen Knoten erfolgen. Dabei wird zuerst der Suchpfad zum Vater des zu löschenden Blattknotens markiert. Befindet sich in diesem Knoten noch ein Punktwert, so wird dieser in den verbleibenden Nachfolgeknoten verschoben. Erst danach erfolgt das Löschen der Knoten. Die neue Kante wird zunächst rot dargestellt.



Nach dem Aufziehen eines Anfragehalbstreifens werden die im Baum besuchten Pfade grün gekennzeichnet. Die bei der Anfrage gefundenen Punkte werden in der gleichen Farbe markiert.

Abbildung 92: Anfrage durchführen

10.7.2 Das Menu Datei zum Prioritätssuchbaum

Neu	Das Programm wird in den Startzustand zurückgesetzt. Ggf. geöffnete Dateien werden geschlossen, Dateinamen gelöscht und die ausgewählte Strukturvariante zurückgesetzt. Die Menüpunkte unter Optionen werden ebenfalls auf die Startwerte zurückgesetzt. In der Titelleiste erscheint die Meldung „**Der Prioritätssuchbaum**“. Alle in diesem Modus nicht nutzbaren Menüpunkte und die Buttons werden deaktiviert.
Öffnen	Beim ersten Mal wird das Dateiverzeichnis **`VisualGeom`** geöffnet, sonst das aktuell eingestellte. Die ausgewählte Strukturvariante bestimmt den Filter für die Dateianzeige. • **`*.pr`** Dateien mit Punkten, die sich in der x-Koordinate voneinander unterscheiden • **`*.pra`** Dateien mit allgemeiner Punktmenge

Tabelle 14: Prio-Menüpunkte des Datei-Menüs (Teil 1)

Speichern	Die abgespeicherte Punktliste wird aktualisiert.
Speichern unter	Der Benutzer hat die Möglichkeit, die Punktliste als Textdatei auf der Festplatte in einem auswählbaren Verzeichnis zu sichern. Bei erstmaligem Speichern wird das Verzeichnis **`Visualgeom`** geöffnet, sonst das aktuell eingestellte Verzeichnis, und nach einem Dateinamen gefragt. Die Dateiendung **`pr`** bzw. **`pra`** wird automatisch als Suffix angehängt.
Drucken	Der Inhalt von Ein- und Ausgabefenster werden auf einem Drucker ausgegeben.
Schließen	Die Bearbeitung des Suchbaums wird beendet und die Fenster geschlossen. Wurde der Prioritätssuchbaum aus dem Startmenü aufgerufen, so wird dieses angezeigt. Andernfalls wird die Applikation oder das Applet beendet.
Beenden	Das Arbeiten mit dem Programm **`VisualGeom`** wird beendet.

Tabelle 15: Prio-Menüpunkte des Datei-Menüs (Teil 2)

10.7.3 Das Menü Bearbeiten zum Prioritätssuchbaum

Alle Punkte löschen	Alle Punkte werden aus der Punktliste entfernt, die Punkte aus dem Eingabereich und der Prioritätssuchbaum aus dem Ausgabebereich gelöscht. Der Benutzer kann die Punkte erneut eingeben. Wurde bereits die Punktliste in einer Datei gespeichert, bleibt der Dateiname erhalten, auch die Einstellungen unter „Optionen“ und „Art des Zeichnens“.
Rasterabstand	Der Abstand zwischen den Rasterlinien zum „Einfangen“ der Punkte ist einstellbar. Auswählbar sind 10, 5 und 2 Bildpunkte Abstand zwischen den ganzzahligen x- bzw. y-Koordinaten. Liegt der Abstand bei 2, so werden keine Rasterlinien mehr angezeigt.
Anzahl Blätter	Bei der Strukturvariante „semidynamisch“ ist die Anzahl der Blätter im Baumskelett einstellbar. Ausgewählt werden können 8, 16, 32 Blätter oder eine beliebige Blattzahl zwischen 2 und 32.
Neuzeichnen	Die Inhalte des Eingabe- und des Ausgabefensters werden gelöscht. Abhängig von der ausgewählten Strukturvariante und der Einstellung unter „Optionen – Art des Zeichnens“ erfolgt anhand der Punktliste der Neuaufbau des Suchbaums. Dabei wird jeweils nur die aktuelle Punktliste genutzt. Gelöschte Punkte haben keinen Einfluss.

Tabelle 16: Prio-Menüpunkte des Bearbeiten-Menüs

10.7.4 Das Menü Strukturvariante zum Prioritätssuchbaum

War bereits eine der Strukturvarianten ausgewählt und wird diese erneut oder eine andere gewählt, so werden Punktliste, Punkte und Suchbaum gelöscht. Welche Strukturvariante gerade ausgewählt ist, kann der Titelleiste entnommen werden.

semidynamisch statisch	Zunächst müssen alle gewünschten Punkte im Eingabebereich angeklickt werden. Dann kann mit Betätigen des Buttons „Baum zeichnen“ der Aufbau des Prioritätssuchbaums im Ausgabebereich veranlasst werden. Nach Änderungen an der Punktmenge muss erneut der Button „Baum zeichnen“ betätigt werden, um den Aufbau des neuen Prioritätssuchbaums zu erhalten.
semidynamisch	Nach der Auswahl dieses Menüpunktes wird ein Skelettbaum mit 16 Blättern gezeichnet. Die Anzahl der Blätter kann über „Bearbeiten-Anzahl Blätter“ geändert werden. Das Anklicken eines Punktes im Eingabebereich bewirkt das Eintragen dieses Punktes im Skelett des Prioritätssuchbaums. Wird versucht einen Punkt mit einer x-Koordinate außerhalb des zugelassenen Intervalls auszuwählen, erfolgt in der Statuszeile die Ausgabe einer Fehlermeldung. Das Löschen eines Punktes führt ebenfalls sofort zum Entfernen des Punktes aus dem Skelettbaum. Wird zu einem Baum mit einer kleineren Knotenanzahl gewechselt und existieren Punkte im Baum mit größeren x-Koordinaten, so werden diese Punkte beim Wechsel gelöscht.

Tabelle 17: Prio-Menüpunkte des Menüs Strukturvariante (Teil 1)

dynamisch	Bei Auswahl dieser Strukturvariante wird zunächst der initiale ∞-Knoten im Ausgabebereich dargestellt. Er enthält einen Splitwert, der größer als alle möglichen x-Koordinaten ist, damit der Löschalgorithmus ohne Sonderfälle auskommt. Die Eingabe eines Punktes bewirkt unmittelbar eine Erweiterung des Prioritätssuchbaums um einen Blattknoten mit dem Wert der x-Koordinate des neuen Punktes. Anschließend wird der Punkt in den erweiterten Suchbaum eingefügt. Wird ein Punkt aus dem Eingabebereich gelöscht, so erfolgt dies unmittelbar auch im Suchbaum. Zusätzlich wird auch das zugeordnete Knotenpaar im Baum gelöscht.
dynamisch balanciert	Diese Variante arbeitet wie die dynamische. Sollten Einfüge- oder Löschvorgänge zu einem unausgewogenen Prioritätssuchbaum führen, erfolgt abschließend eine Balancierung nach den AVL-Kriterien.

Tabelle 18: Prio-Menüpunkte des Menüs Strukturvariante (Teil 2)

10.7.5 Das Menü Optionen zum Prioritätssuchbaum

Punktmenge	Abhängig von der ausgewählten Strukturvariante besteht die Möglichkeit, zwischen disjunkter Punktmenge und allgemeiner Punktmenge zu wechseln.
Grafik in Datei	Ein Abbild vom „Eingabebereich“ oder „Ausgabebereich“ kann als Datei im JPEG-Format auf der Festplatte abgespeichert werden, um es später mit einem Textverarbeitungsprogramm oder mit einem Grafikprogramm weiter zu bearbeiten. Dateiname und Verzeichnis sind frei wählbar. Das Suffix `jpg` wird automatisch gesetzt.
Art des Zeichnens	Hier kann zwischen „animiert“ und „sofortiger Aufbau“ gewählt werden. Welche dieser Optionen aktiviert ist, hängt von der jeweils ausgewählten Strukturvariante ab.

Tabelle 19: Prio-Menüpunkte des Menüs Optionen

10.7.6 Das Menü Hilfe zum Prioritätssuchbaum

Hilfethemen	Hier kann die Online-Anleitung zur Benutzung von **`VisualGeom`** aufgerufen werden. Über das Inhalts- und das Indexverzeichnis lässt sich dabei gezielt auf eine bestimmte Hilfedatei zugreifen. Die Modifikation der einzelnen Hilfedateien ist problemlos möglich, da es sich um HTML-Dateien handelt.
Demos	Es kann zwischen Demos zum semidynamischen und dynamischen Prioritätssuchbaum gewählt werden. • Semidynamischer Prioritätssuchbaum Zuerst erfolgt der Aufbau eines Baumgerüstes mit 16 Blättern. In dieses werden dann nacheinander Punkte eingefügt und anschließend einige wieder gelöscht. Abschließend wird ein Anfragehalbstreifen eingezeichnet. Alle bei der Bereichsanfrage besuchten Knoten und die gefundenen Punkte werden markiert. • Dynamischer Prioritätssuchbaum Gleichzeitig mit der Eingabe der Punkte wird der Prioritätssuchbaum aufgebaut. Beim Löschen eines Punktes wird auch das zugeordnete Knotenpaar aus dem Prioritätssuchbaum entfernt. Abschließend wird ein Anfragehalbstreifen eingezeichnet. Alle bei der Bereichsanfrage besuchten Knoten und die gefundenen Punkte werden markiert. • Dynamisch balancierter Prioritätssuchbaum Gleichzeitig mit der Eingabe der Punkte wird der Prioritätssuchbaum aufgebaut. Entsteht ein unausgewogener Prioritätssuchbaum, wird er zunächst balanciert, bevor weitere Punkte eingefügt werden. Abschließend wird ein Anfragehalbstreifen eingezeichnet. Alle bei der Bereichsanfrage besuchten Knoten und die gefundenen Punkte werden markiert.

Tabelle 20: Prio-Menüpunkte des Hilfe-Menüs

11. Schlussfolgerungen und Ausblick

In der Elaboration-Phase konnten wir keine Umfrage bei den zukünftigen Benutzern von **VisualGeom** durchführen, um deren Wünsche und Anregungen für die Programmentwicklung zu berücksichtigen. Deshalb haben wir nur unsere eigenen Anforderungen an das Softwareprodukt einbringen können.
Die Validierung von **VisualGeom** ließe sich in [FU1840] im Sommersemester 2000 nachholen. Umsetzbare Wünsche der Benutzer könnten in einer nächsten Release aufgenommen werden, um das Programm noch besser auf die Bedürfnisse der Benutzer anzupassen.

Bei der Programmierung bereiteten uns die Umstellung auf die objektorientierte Programmiersprache Java und die häufigen Erweiterungen der Firma SUN einige Schwierigkeiten. Der Umgang mit einem derart mächtigen Software Development Kit führte dazu, dass wir sicher nicht die Resourcen der Sprache ausschöpfen konnten. Hierzu fehlte uns die praktische Erfahrung. Umfangreichere Vorkenntnisse hätten insbesondere im Animationsbereich noch ansprechendere Lösungen möglich gemacht.

Im einführenden Kapitel 6 zum Rational Unified Process werden die Entwicklungsschritte für umfangreiche Softwareprojekte beschrieben. Da es sich bei der Entstehung von **VisualGeom** um ein recht kleines Softwareprodukt handelt, haben wir uns nur an dem Prinzip der Anwendungsfallzentrierung und der iterativen Entwicklung orientiert. In ausgewählten Beispielen haben wir die UML-Dokumentationsformen des Konzeptes genutzt, um Zusammenhänge zu verdeutlichen.

Für eine der nächsten Releases von **VisualGeom** erscheinen uns folgende Punkte sinnvolle Ergänzungen darzustellen:

- Applikation und Applet werden um die Datenstruktur Bereichssuchbaum erweitert.
- Die Suchbäume lassen sich kollabieren, um so auch sehr umfangreiche Bäume zu visualisieren.
- Zusätzlich zur Menüleiste wird eine Symbolleiste für häufig benutzte Menüpunkte eingefügt.

- Für die Punktmengen wird ein Datenformat festgelegt, das es ermöglicht, diese Datei unmittelbar als Textdatei in einem Textverarbeitungsprogramm in der Form {(x1|x2), ...(xn|yn)} einzulesen. Damit auch das externe Erstellen von Punktlisten mit einem Texteditor möglich ist, wird die Methode zum Lesen der Punktliste um einen Syntaxchecker erweitert.
- Die Druckmethode wird abhängig von der Geometrie des jeweiligen Suchbaums optimiert. Dabei erscheint es aktuell sinnvoll auf die von SUN implementierte Methode zu verzichten und eine eigene oder aus dem Internet erhältliche Druckroutine zu verwenden.
- Da in der hier vorliegende Release das Speichern von Eingabe- und Ausgabebereich nur im JPEG-Format möglich ist, bietet sich der Einsatz des inzwischen von SUN erhältlichen Java-Advanced-Imaging-Paketes (JAI) an. Hiermit kann dann der Benutzer beim Speichern zwischen den verschiedenen Grafikformaten selbst auswählen.
- Die Animation sollte auch auf die Balancierung der Bäume erweitert werden, um die damit verbundenen komplexen Vorgänge besser zu visualisieren.

Abschließend wollen wir die mit dieser Arbeit für uns gewonnene Erkenntnis darlegen.

> Die Bedeutung der Aussage von Franklin „Tell me and I forget, Teach me and I remember, Involve me and I learn.“ konnten wir im Rahmen unserer Arbeit selbst erleben, als wir die vom anderen erstellte Datenstruktur testeten.
>
> Durch das enaktive Auseinandersetzen mit der visualsierten Datenstruktur erfuhren wir ein tieferes Verständnis als es beim Bearbeiten von [FU1840, Seiten 118 bis 125 und 130 bis 135] mit Papier und Bleistift möglich war.

Aufteilung der Gruppenarbeit

In Absprache mit dem Lehrgebiet VI und dem Prüfungsamt Mathematik und Informatik wurde die Diplomarbeit “Visualisierung Geometrischer Datenstrukturen unter Berücksichtigung softwareergonomischer und lernpsychologischer Gesichtspunkte” als Gruppenarbeit erstellt. Die Abgrenzung der Themen in der Arbeit und der Programmteile zwischen Ingrid und Klaus-Dieter Schumacher erfolgte gemäß der nachfolgenden Tabelle.

	Ingrid Schumacher	**Klaus-Dieter Schumacher**
Programmcode	• Prioritätssuchbaum ohne Drucken, Dateiverwaltung und Hilfesystem • Benutzeroberfläche • Menüklassen • Implementation als Applet • Blattsuchbaum • Implementierung des Algorithmus für das schöner Zeichnen	• 2-d-Baum ohne Menüklassen, Applet, Blattsuchbaum und den Algorithmus für das schöner Zeichnen • Methoden zum Laden und Speichern der Punktliste • Methoden zum Drucken von Eingabe- und Ausgabebereich • Methoden zum Speichern von Eingabe- und Ausgabebereich als JPEG-Datei • Methoden zum Hilfesystem, hs-, XML-, Index- und Hilfedateien
Kapitel der Diplomarbeit	1, 2, 7.5, 8.2.1, 8.2.2, 8.4, 9.2, 10.7	3, 4, 5, 7.4, 8.2.4 bis 8.2.7, 8.3, 9.1, 10.6, A2

Alle nicht genannten Kapitel der Diplomarbeit und des erstellten Programmcodes lassen sich nicht unmittelbar einem von uns beiden zuordnen.

Erklärung

Hiermit versichern wir, dass wir die Diplomarbeit selbständig mit der unter "Aufteilung der Gruppenarbeit" genannten Abgrenzung verfasst haben. Auch haben wir keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.
Zitate sind als solche in der Arbeit kenntlich gemacht worden.

Stockelsdorf, im November 1999

Ingrid Schumacher Klaus-Dieter Schumacher

A N H A N G

I N H A L T S V E R Z E I CH N I S

A.1 LITERATURVERZEICHNIS 1

A.1.1 BÜCHER UND ZEITSCHRIFTEN 1
A.1.2 ONLINE-DOKUMENTATIONEN/BÜCHER/TUTORIAL 4
A.1.3 NEWS-GRUPPEN 5
A.1.4 DOWNLOADS 5

A.2 ANLEITUNG ZUM ERSTELLEN DER HILFEDATEIEN AN EINEM BEISPIEL 7

A.2.1 ALLGEMEINE HINWEISE 7
A.2.2 DIE DATEI **KDHELP.HS** 8
A.2.2 DIE DATEI **MAP.JHM** 11
A.2.3 DIE DATEI **KDHELPTOC.XML** 12
A.2.4 DIE DATEI **KDHELPINDEX.XML** 13

A.3 VERWENDETE UML-SYMBOLE 15

A.4 ANMERKUNGEN ZUR PLATTFORMUNABHÄNGIGKEIT 18

A.5 CD 21

Verzeichnis der Abbildungen

Abbildung A1: Beispiel für die Aufteilung der Verzeichnisse 7
Abbildung A2: **KDHelp.hs** 9
Abbildung A3: Die Datei **Map.jhm** 11
Abbildung A4: Auszug aus einer TOC-Datei 12
Abbildung A5: Auszug aus der Datei **KDHelpIndex.xml** 13

Verzeichnis der Tabellen

Tabelle A1: Liste der Download-WWW-Adressen (Teil 1) 5
Tabelle A2 Liste der Download-WWW-Adressen (Teil 2) 6
Tabelle A3: Tags in der HelpSet-Datei **Help.hs** 10
Tabelle A4: Bedeutung der Tags in der Map-Datei 11
Tabelle A5: Tags in der TOC-Datei (Teil 1) 12
Tabelle A6: Tags in der TOC-Datei (Teil 2) 13
Tabelle A7: Tags in der Datei **KDHelpIndex.xml** 14
Tabelle A8: Verwendete Use-Case-Diagrammsymbole 15
Tabelle A9: Verwendete Aktivitätsdiagrammsymbole 16
Tabelle A10: Verwendete Klassendiagrammsymbole 16
Tabelle A11: Verwendete Interaktionsdiagrammsymbole 17

A.1 Literaturverzeichnis

A.1.1 Bücher und Zeitschriften

[BOOC99] Booch, Jacobson, Rumbaugh
The Unified Software Development Process
Addison-Wesley; Bonn; 1999

[BOOC99a] Booch, Jacobson, Rumbaugh
Das UML-Benutzerhandbuch
Addison-Wesley; Bonn; 1999

[BURK97] Burkhardt, Rainer
UML - Unified Modeling Language; Objektorientierte Modellierung für die Praxis
Addison-Wesley; Bonn; 1997

[CORN97] Cornell, Gary und Horstmann, Cay S.
Core Java 1.1: Vol. 1 – Fundamentals, Vol. 2 - Advanced Features
Sunsoft Press; Prentice Hall; 1997

[DEIN96] Deininger, M. u. a.
Studien-Arbeiten
Teubner; Stuttgart; 3. Aufl. 1996

[DUDE88] Claus, V. und Schwill, A.
Duden, Informatik
Dudenverlag; Mannheim; 1988

[FU1663] Kurs 1663 der FernUniversität Hagen
Datenstrukturen
FernUniversität, Gesamthochschule Hagen 1999

[FU1840] Kurs 1840 der FernUniversität Hagen
Algorithmische Geometrie
FernUniversität, Gesamthochschule Hagen 1996

[FU4763] Kurs 4763 der FernUniversität Hagen
Software-Ergonomie und Neue Techniken

[GAMM98] Gamma, Erich u. a.
Entwurfsmuster; Elemente wiederverwendbarer objektorientierter Software
Addison-Wesley; Bonn; 1998; 2. Nachdruck

[HAM92] Hampe-Neteler, u.a
Software-Ergonomie - Verfahren der Evaluierung und Standards zur Entwicklung von Benutzeroberflächen
Forschungsbericht des Studiengangs Informatik der Uni Bremen
Bericht 2/92; 1992

[HERC94] Herczeg, M.
Software-Ergonomie; Grundlagen der Mensch-Computer-Kommunikation
Addison-Wesley; Bonn; 1994

[HETT80] Hettinger, Th u. a.
Ergonomie am Arbeitsplatz
Kiel Verlag; 1980

[IS9922] Noack, J. und Schienmann, B.
Objektorientierte Vorgehensmodelle im Vergleich
in: Informatik-Spektrum Band 22, Heft 3, Juni 1999, Seite 166 bis 179

[KOCH91] Koch, M. u. a.
Software-Ergonomie
Springer; Wien; 1991

[MEH94a] Mehlhorn, K.
Data Structures and Algorithms 1: Sorting and Searching
Springer-Verlag; Berlin; 1984

[MEH94b] Mehlhorn, K.
Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry
Springer-Verlag; Berlin; 1984

[OEST98] Oesterreich, Bernd
Objektorientierte Softwareentwicklung
Oldenburg Verlag; München; 1998; 4. Auflage

[OPPE94] Opperman, R u. a.
Einführung in die Software-Ergonomie
in Eberleh, H u. a.; Software-ergonomische Evaluation
deGruyter; 1994

[OTTM93] Ottmann, Thomas und Widmayer, Peter
Algorithmen und Datenstrukturen
B I; Mannheim; 1993; 2. Auflage

[REIN81] Reingold, E. M. und Tilford, J. S.
Tidier Drawings of Trees
In: IEEE Transactions on Software Engineering, Vol. SE-7, No. 2, March 1981, Page 223ff.

[RICH89] Richards, Jack und Rogers, Theodore
Approaches and Methods in Language Teaching; Seite 100
Cambridge; 1989

[ROWE98] Rowe, G. W.
An Indruction to Data Structures and Algorithms with Java
Prentice Hall Europe; Munich; 1998

[RUMB94] Rumbaugh, J.
Objektorientiertes Modellieren und Entwerfen
Hanser; München; 1994

[SELLE97] Selle; Norbert
JavaAnimation Style Guides; Version 1.1 vom 10.08.1997
Internes Dokument des Lehrgebiets Praktische Informatik VI der FernUniversität

[STAM84] Stampe, E.
Repetitorium Fachdidaktik Mathematik
Klinkhardt; Bad Heilbrunn; 1994

[VEST78] Vester, F.
Denken, Lernen, Vergessen
dtv; Stuttgart; 1978

[WAND93] Wandmacher, J.
Software-Ergonomie
DeGruyter; Berlin; 1993

[WETH79] Wetherell, C. und Shannon, A.
Tidy Drawings of Trees
In: IEEE Transactions on Software Engineering
Vol. SE-5, No. 5, September 1979, Page 514ff.

A.1.2 Online-Dokumentationen/Bücher/Tutorial

[SUJT99] http://www.javasoft.com/docs/books/tutorial/index.html
Online version of The Java Tutorial

[PART99] http://www.boku.ac.at/javaeinf/
Java-Einführung von Hubert Partl (BOKU Wien)

[HOLU98] http://www.javaworld.com/javawolrd/jw-09-1998/jw-09-threads.html
Allen Holub
Programming Java threads in the real world, Part 1

A.1.3 News-Gruppen

Während der Programmerstellung wurden von uns die Nachrichten in folgenden Gruppen als hilfreiche Hinweise für die Programmerstellung, das Vorhandensein von Bugs oder den Verweis auf spezielle Methoden genutzt:

- comp.lang.java.gui
- comp.lang.java.help
- comp.lang.java.programmer
- de.comp.lang.java

A.1.4 Downloads

Produkt	Internetquelle
JDK2 für • Windows 95 • Windows 98 • Windows NT	http://java.sun.com/products/jdk/1.2/download-windows.html
JRE2 für • Windows 95 • Windows 98 • Windows NT	http://java.sun.com/products/jdk/1.2/jre/download-windows.html **Hinweis**: Dieses Produkt schließt das Plug-In für die Browser Netscape Communicator und Internet Explorer unter Windows ein.
JDK2 für Linux	http://www.blackdown.org/java-linux/mirrors.html
JRE2 für Linux	http://www.blackdown.org/java-linux/mirrors.html
JDK2 für Solaris	http://www.sun.com/solaris/jdk/download.1.2.1_03/
JRE2 für Solaris	http://www.sun.com/software/solaris/jre/index.html
JRE2-Plug-In für Solaris	http://www.sun.com/solaris/netscape/jpis/index.html

Tabelle A1: Liste der Download-WWW-Adressen (Teil 1)

Produkt	Internetquelle
JavaHelp	http://java.sun.com/products/javahelp/index.html
HTMLCONF12.ZIP	http://java.sun.com/products/plugin/1.2/jinstall-12-win32.cab **Hinweis**: Dies ist das Java-Programm, um JDK2-Applets durch Anpassen der HTML-Startseite mit dem Netscape Communicator oder dem Internet Explorer starten zu können.
Kawa	http://www.tek-tools.com/kawa **Hinweis**: Integrierte Entwicklungsumgebung unter Windows
Rational Rose	http://www.rational.com/rose **Hinweis**: [BOOC99] liegt ein Bestellzettel für eine CD mit der kostenlosen Evaluation Version dieses Produktes bei.

Tabelle A2 Liste der Download-WWW-Adressen (Teil 2)

A.2 Anleitung zum Erstellen der Hilfedateien an einem Beispiel

A.2.1 Allgemeine Hinweise

Zur besseren Übersichtlichkeit ist es empfehlenswert, das Hilfesystem in ein eigenes Unterverzeichnis zu packen. Dies sollte für die Images und die Hilfedateien ebenfalls erfolgen.

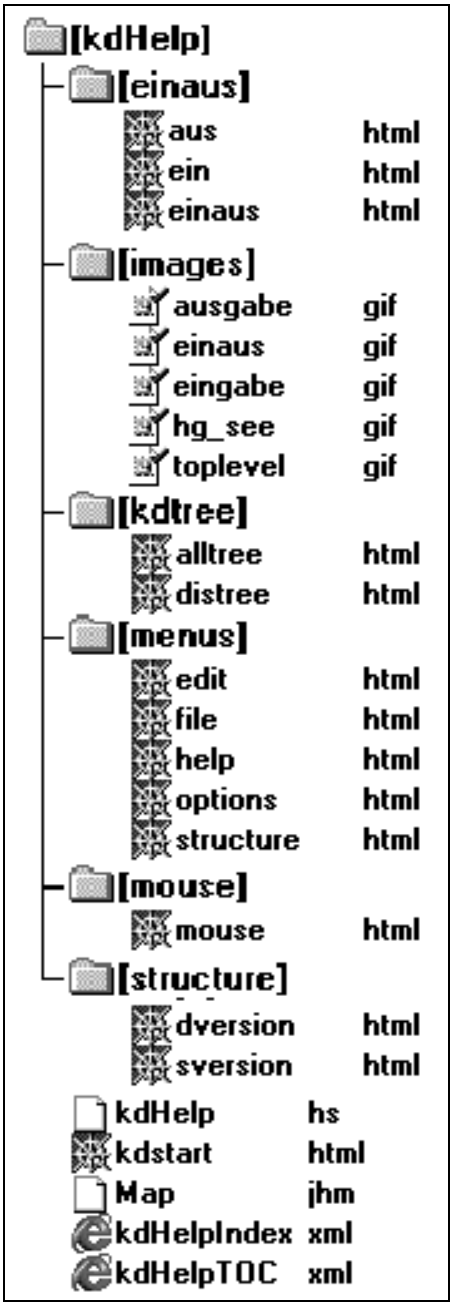


Abbildung A1: Beispiel für die Aufteilung der Verzeichnisse

Zunächst werden mit einem geeigneten Editor (z. B. Netscape Composer, Frontpage) die HTML-Dateien mit den Hilfetexten erstellt. Diese werden, um die Übersichtlichkeit sicherzustellen, geeignet in verschiedenen Unterverzeichnissen abgelegt (Abbildung A1).

Sind diese Dateien fertig, müssen sie mit dem Hilfesystem verknüpft werden. Wie der Quellcode in Abbildung 54 zeigt, wird der Applikation der Name der HelpSet-Datei bekannt gegeben. Dies ist erforderlich, da eine Applikation verschiedene HelpSet-Dateien nutzen kann.
Es müssen nun folgende Datendateien mit einem Texteditor erstellt werden. Sie sind alle im Hilfeverzeichnis, hier **`kdHelp`**, zu speichern:

- Die HelpSet-Datei **`kdHelp.hs`**
- Die Metadatei zur Beschreibung der Struktur des Hilfesystems **`kdHelpTOC.xml`**
- Die Indexdatei für den gezielten Zugriff auf einzelne Suchbegriffe **`kdHelpIndex.xml`**
- Die Datei für das Mapping von IDs und URLs **`Map.jhm`**
- Die HTML-Datei, deren Inhalt beim Aufruf des Hilfesystem angezeigt wird **`kdStart.htlm`**

A.2.2 Die Datei `kdHelp.hs`

Der nachfolgende Auszug aus Abbildung A2 verdeutlicht den grundsätzlichen Aufbau einer HelpSet-Datei.

```
<?xml version='1.0' encoding='ISO-8859-1'
?>
<!DOCTYPE helpset>


<helpset version="1.0">


  <!-- title -->
  <title>Hilfe zum Kd-Suchbaum</title>
  <!-- maps -->
  <maps>
     <homeID>top</homeID>
     <mapref location="Map.jhm"/>
  </maps>


  <!-- views -->
  <view>
    <name>TOC</name>
    <label>Inhaltsverzeichnis</label>
    <type>javax.help.TOCView</type>
    <data>kdHelpTOC.xml</data>
  </view>


  <view>
    <name>Index</name>
    <label>Indexverzeichnis</label>
    <type>javax.help.IndexView</type>
    <data>kdHelpIndex.xml</data>
  </view>


</helpset>
```

Abbildung A2: **kdHelp.hs**

In dieser Datei werden verschiedene Tags verwendet. Ihre Bedeutung ist der Tabelle A3 entnehmbar.

Tag	Bedeutung
<helpset>	Mit diesem Tag wird JavaHelp mitgeteilt, dass es sich bei dieser Datei um eine HelpSet-Datei handelt.
<title>	Der Name des HelpSets wird festgelegt.
<maps>	Angabe des Namens der zu nutzenden den Map-Datei
<homeID>	Diese Standard-ID wird angezeigt bei Aufruf der HelpViewer, ohne dass dabei eine ID genannt wird.
<mapref location="...">	Der Name der Map-Datei wird bestimmt.
<view>	Navigationsmöglichkeiten wie die Angabe des Inhalts- und das Indexverzeichnises werden beschrieben.
<name>	Der Name des Navigators (TOC, Index, Search) wird angegeben.
<label>	Es erfogt das Festlegen des Namens für den Benutzer.
<type>	Dieses Tag schließt die Angabe des Pfades der zu verwendenden Navigatorklasse ein. Hier z. B.: **`javax.help.TOCView`**.
<data>	Dieser Tag gibt den Pfad auf die für das Inhaltsverzeichnis oder das Indexverzeichnis verwendeten Datendateien an. Für die Suchmaschine ergibt sich folgende spezielle Beschreibung: **`<data`** **`engine="com.sun.java.help.search.DefaultSearchEngine">`** **`JavaHelpSearch`** **`</data>`**

Tabelle A3: Tags in der HelpSet-Datei `Help.hs`

A.2.2 Die Datei `Map.jhm`

Der nachfolgende Auszug von Abbildung A3 verdeutlicht den grundsätzlichen Aufbau einer Map-Datei.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE map>

<map version="1.0">
      <mapID target="toplevelfolder"   url="images/toplevel.gif" />
      <mapID target="top"              url="kdstart.html" />
      <mapID target="kd.intro"         url="kdstart.html" />

      <mapID target="menus.file"       url="menus/file.html" />
      <mapID target="menus.edit"       url="menus/edit.html" />
      <mapID target="menus.structure"  url="menus/structure.html" />
      <mapID target="menus.options"    url="menus/options.html" />
      <mapID target="menus.help"       url="menus/help.html" />
      <mapID target="menus.help"       url="menus/help.html" />

      ...
</map>
```

Abbildung A3: Die Datei `Map.jhm`

An der tabellarischen Darstellung fällt bereits auf, dass diese Datei in der linken Spalte die jeweiligen IDs und in der rechten Spalte die zugeordneten URLs enthält.

Tag	Bedeutung
<map>	Mit diesem Tag beginnt eine Tabelle von Zuordnungen zwischen den IDs und den URLs.
<mapID target="..." url="..."/>	Die Zuordnung zwischen der URL und der in den nachfolgenden Dateien verwendeten ID wird beschrieben. Die Angabe der URL ist relativ, da der absolute Pfad auf das Hilfesystem bereits in der Applikation festgelegt wurde.

Tabelle A4: Bedeutung der Tags in der Map-Datei

A.2.3 Die Datei kdHelpTOC.xml

Der nachfolgende Auszug verdeutlicht den grundsätzlichen Aufbau einer HelpTOC-Datei.

```
<?xml version='1.0' encoding='ISO-8859-1'  ?>
<!DOCTYPE toc>
<toc xml:lang="de">
<tocitem text="2-d-Baum" image="toplevelfolder">

   <tocitem text="Menüs">
      <tocitem text="Menü &quot;Datei&quot;"            target="menus.file"/>
      <tocitem text="Menü &quot;Bearbeiten&quot;"       target="menus.edit"/>
      <tocitem text="Menü &quot;Strukturvariante&quot;" target="menus.structure"/>
      <tocitem text="Menü &quot;Optionen&quot;"         target="menus.options"/>
      <tocitem text="Menü &quot;Hilfe&quot;"            target="menus.help"/>
   </tocitem>

   <tocitem text="Mausaktionen"                        target="mouse.mouse"/>

   ...

</tocitem>
</toc>
```

Abbildung A4: Auszug aus einer TOC-Datei

Diese Datei ermöglicht den Aufbau eines Inhaltsverzeichnisses mit Untergliederungspunkten.

Tag	Bedeutung
<toc>	Diese so spezifizierte Datei wird als eine Inhaltsverzeichnisdatei interpretiert.
<tocitem text="..." image="...">	Ein Verzeichniseintrag mit dem unter **text** festgelegten Namen wird definiert. Dieser Name wird bei der Auswahl des Eintrags angezeigt. Mit **image** wird das zusammen mit dem Eintrag auszugebende Bild angegeben. Dies erfolgt mittels einer ID. Diese verweist auf die URL einer GIF- oder JPEG-Datei.

Tabelle A5: Tags in der TOC-Datei (Teil 1)

<tocitem text="...">	ein Verzeichniseintrag mit dem unter **`text`** festgelegten Namen wird definiert. Dieser Name wird bei der Auswahl des Eintrags angezeigt. Dieser Eintrag dient als übergeordneter Gliederungspunkt / Zusammenfassung von Hilfedateien (s. oben: **`text="Menüs"`**). Beim Anklicken wird das Menü auf- bzw. zugeklappt.
<tocitem text="..." target="...">	Es wird ein Verzeichniseintrag mit dem unter **`text`** festgelegten Namen definiert. Dieser Name wird bei der Auswahl des Eintrags angezeigt. Mit **`target`** wird die ID einer Datei zugeordnet, die bei Auswahl dieses Menüpunktes angezeigt werden soll.

Tabelle A6: Tags in der TOC-Datei (Teil 2)

A.2.4 Die Datei `kdHelpIndex.xml`

Der nachfolgende Auszug verdeutlicht den grundsätzlichen Aufbau einer HelpIndex-Datei.

```
<?xml version='1.0' encoding='ISO-8859-1'  ?>
<!DOCTYPE index>

<index version="1.0">

   <indexitem text="Ausgabebereich">
      <indexitem text="Ausgabebereich"       target="einaus.aus"/>
      <indexitem text="Ein-Ausgabebereich"   target="einaus.einaus"/>
   </indexitem>

   <indexitem text="Eingabebereich"          target="einaus.ein"/>

   ...
</index>
```

Abbildung A5: Auszug aus der Datei `kdHelpIndex.xml`

Tag	Bedeutung
<index>	Für das Hilfesystem wird kenntlich gemacht, dass es sich um eine Indexdatei handelt.
<indexitem text="...">	Der durch **`text`** spezifizierte Indexwert wird dargestellt. Er dient als Oberbegriff, zu dem eingerückt die weiteren Suchbegriffe dargestellt werden.
<indexitem text="..." target="..."/>	Es wird der durch **`text`** spezifizierte Indexwert dargestellt. Mit **`target`** wird die ID einer Datei zugeordnet, die bei Auswahl dieses Menüpunktes angezeigt werden soll.

Tabelle A7: Tags in der Datei `kdHelpIndex.xml`

A.3 Verwendete UML-Symbole

In unserer Diplomarbeit haben wir zur Veranschaulichung der Zusammenhänge Use-Case-, Aktivitäts-, Interaktions- und Klassendiagramme genutzt. Hier folgen nun die Erläuterungen zu den dabei jeweils verwendeten Symbolen.
Weitergehende Informationen sind u. a. in [BOOC99a], [BURK97] und [OEST98] u finden.

Use-Case-Diagramm

„Ein Anwendungsfall beschreibt eine Menge von Aktivitäten eines Systems aus der Sicht seiner Akteure, die für die Akteure zu einem wahrnehmbaren Ergebnis führen. Ein Anwendungsfall wird stets durch einen Akteur initiiert. Ein Anwendungsfall ist ansonsten eine komplette, unteilbare Beschreibung.“ [OEST98, Seite 207]

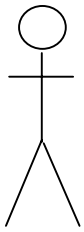
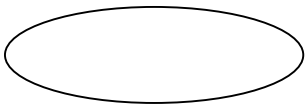
Symbol	Erläuterung
	Mit diesem Symbol wird der Akteur gekennzeichnet. Hierbei handelt es sich um die am Anwendungsfall beteiligten Personen. Unter dem Symbol ist die Symbolbezeichnung zu platzieren.
	Hierbei handelt es sich um das Symbol für den Anwendungsfall. Der Name des Anwendungsfalls wird in das Symbol eingetragen.
	Mit der Verbindungslinie wird kenntlich gemacht, welche Anwendungsfälle dem jeweiligen Akteur zu zuordnen sind.

Tabelle A8: Verwendete Use-Case-Diagrammsymbole

Aktivitätsdiagramm

Ein Aktivitätsdiagramm dient der Veranschaulichung des Flusses von einer Aktivität zu anderen innerhalb eines Systems. Damit ergibt sich:

„Eine Aktivität ist ein Zustand mit einer internen Aktion und einer oder mehreren ausgehenden Transaktionen, die automatisch dem Abschluss der internen Aktion folgen. Eine Aktivität ist ein einzelner Schritt in einem Ablauf. Eine Aktivität kann mehrere ausgehende Transaktionen haben, wenn diese durch Bedingungen unterschieden werden können.“ [OEST99, Seite 295]

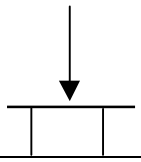
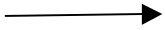
Symbol	Erläuterung
	Mit diesem Symbol wird eine Aktivität visualisiert. Eingetragen wird in dieses Symbol eine Aktionsbeschreibung, z. B. in frei formulierter Form.
	Dieses Symbol macht kenntlich, dass die Transaktion aufgeteilt wird in zwei parallel laufende Teiltransaktionen.
	Angabe der Flussrichtung

Tabelle A9: Verwendete Aktivitätsdiagrammsymbole

Klassendiagramm

In einem Klassendiagramm werden Klassen und die zwischen ihnen bestehenden Beziehungen veranschaulicht.

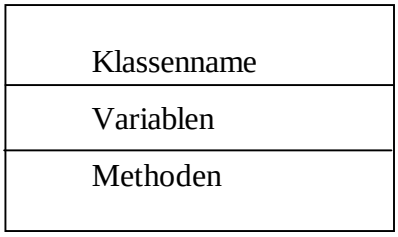
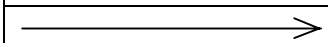
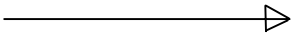
Symbol	Erläuterung
	Eine Klasse wird im Klassendiagramm durch ein Rechteck mit eingetragenem Klassennamen veranschaulicht. Des weiteren können die Variablen, die Konstruktoren und die Methoden in dem Symbol eingetragen werden.
	Der geöffnete Pfeil macht eine Benutzungsbeziehung kenntlich.
	Mit dem geschlossenen Pfeil ist die Information verbunden, dass es sich um eine Vererbungsbeziehung handelt. Dabei weist der Pfeil auf die Oberklasse, von der die Unterklassen erben.

Tabelle A10: Verwendete Klassendiagrammsymbole

Interaktionsdiagramm

Häufig wird in der Fachliteratur der Begriff Interaktionsdiagramm als Sammelbegriff für Sequenz-, Kollaborations- und Aktivitätsdiagramm verwendet. Wir verwenden hier den Begriff synonym zum Begriff Sequenzdiagramm.

„Eine Sequenz zeigt eine Reihe von Nachrichten, die eine ausgewählte Menge von Objekten in einer zeitlich begrenzten Situation austauscht, wobei der zeitliche Ablauf betont wird." [OEST99, Seite 306]

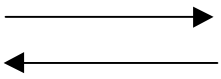
Symbol	Erläuterung
	Die gestrichelte senkrechte Linie macht ein Objekt kenntlich. Sie wird auch als Lebenslinie des betreffenden Objektes bezeichnet.
	In das über der Lebenslinie platzierte Rechteck wird der Name des Objektes eingetragen.
	Dieses auf der Lebenslinie platzierte Symbol kennzeichnet den Steuerungsfokus. Er gibt an, welches Objekt aktuell aktiv ist.
	Die waagerechten Pfeile visualisieren Nachrichten zwischen den durch die Lebenslinien symbolisierten Objekten. Auf einem Pfeil wird die jeweilige Nachricht notiert.
	Mit diesem Symbol über einem Steuerungsfokus verdeutlichen wir, dass das Auslösen der betreffenden Nachricht durch den Akteur verursacht wurde.

Tabelle A11: Verwendete Interaktionsdiagrammsymbole

A.4 Anmerkungen zur Plattformunabhängigkeit

Zum Thema Plattformunabhängigkeit von Java stellt [HOLU98] fest:

> "Unfortunately, Java's promise of platform independence falls flat on is face in the threads arena. ... This isn't really Java's fault: it's almost impossible to write a truly platform-independed threading system."

Dies mussten wir leider wiederholt bei den Programmtests feststellen. Anfangs war Laufzeitverhalten der Animationen unter Windows 95/98, Linux und Solaris recht unterschiedlich.
Es wurden diverse „Klimmzüge" erforderlich, um annähernd vergleichbares Verhalten zu erzielen.
Unter Windows liefen die Animationen bereits bei unseren ersten Programmierversuchen recht flüssig ab, während sie unter Linux und Solaris nicht vertretbar langsam waren.
Zu bedenken bleibt dabei, dass Java 10 Prioritätsebenen für Threads unterstützt, während Solaris 2^{31} und Windows NT nur 7 besitzt. Daran wird schon deutlich, dass die Abbildung der Threads unter den verschiedenen Betriebssysteme verschiedenes Verhalten bewirkt.

Des weiteren lassen sich die Java-Klassen unter Solaris und Linux mit insgesamt vier Kombinationen compilieren.

	JIT	**NOJIT**
Green-Thread	x	x
Native-Thread	x	x

Die Bedeutung der vier Begriffe ist der nachfolgenden tabellelarischen Darstellung zu entnehmen.

Während die JDK 2 Pre-Release 1 der JDK2 Version 1.2 für Linux unter der SuSE-Distribution in der Version 6.1 das System beim Compilieren mit Green-Threads aufhängte, konnten die Klassen mit der Pre-Release 2 nur bei der Kombination Green+JIT nicht compiliert werden. Bei den drei übrigen Varianten zeigte sich ebenso wenig ein unterschiedliches Laufzeitverhalten wie bei der direkten Verwendung der unter Windows compilierten Klassen.

Begriff	**Bedeutung**
Green-Thread	Threading wird durch die VM (virtual maschine) realisiert. "Green Threads versus Native Threads When Java 1.0 first came out on Solaris it did not use the native Solaris library libthread.so to support threads. Instead it used runtime thread support that had been written in Java for an earlier project code named 'Green'. That thread library came to be known as 'green threads'."[85] Es handelt sich also um ein System kooperativer Threads.
Native-Thread	Threading wird durch das Betriebssystem realisiert. Dies macht hier eigentlich nur Sinn, wenn mehrere Prozessoren zur Verfügung stehen.
JIT	Es wird der **J**ust-**I**n-**T**ime-Compiler verwendet.
NOJIT	Es wird der **J**ust-**I**n-**T**ime-Compiler nicht verwendet.

Auch existiert JDK2 auf den drei Betriebssystemplattformen WindowsXX, Solaris und Linux immer noch nicht in der gleichen Version. Auch hierdurch ergaben sich Implementationsprobleme. Diverse kleinere Bugs, die die Versionen 1.2 und 1.2.1 enthalten, wurden mit der Version 1.2.2 unter Windows beseitigt.
Der aktuelle Versionsstand Ende Oktober 1999 ist:

- Windows JDK2 in der Version 1.2.2, JDK2 in der Version 1.3beta
- Solaris JDK2 in der Version 1.2.1_03
- Linux JDK2 Pre-Release 2 für die Version 1.2

Für MAC-OS existiert weiterhin keine JDK2-Version. Also kann unser Programm weiterhin nicht auf dieser Plattform genutzt werden. Da wir neben Swing-Komponenten in unserem Programm auch Komponenten des 2D-API genutzt haben, ist das Compilieren unter JDK1.1 trotz einer für diese Version verfügbaren Swing-Variante nicht möglich.

[85] Dieser Hinweis stammt aus einem Artikel au einer der englischen Java-News-Groups.

Während mit **VisualGeom** als Applikation das Speichern eines Image als JPEG-Datei unter Windows und Linux problemlos möglich war, kam es unter Solaris trotz der JDK2 Version 1.2.1_03 zu folgender Fehlermeldung:

```
Exception occured during event dispatching:
java.lang.IllegalArgumentException:
      NumComponents not in sync with COLOR_ID
at com.sun.codec.jpeg.JPEGCoder.getDefaultJPEGEncoderParam()
```

Diese Fehlermeldung konnte auch durch eine Ausnahmenbehandlung nicht abgefangen werden.

Aktuell ist auch die Internetnutzung nur eingeschränkt möglich, da für das Starten des Applets mit dem Netscape Communicator oder dem Internet Explorer zunächst das JRE2 vom SUN-Server gezogen und installiert werden muss. Des weiteren existiert immer noch kein hierfür erforderliches Plug-In unter Linux. Auch muss die HTML-Startseite mit dem HTML-Converter von SUN für die Nutzung des Plug-Ins vorbereitet werden. Das Ergebnis diese Aufbereitung ist allerdings auch, dass eine derart aufbereitete HTML-Seite nicht mehr mit dem **appletviewer** geladen werden kann. Auch hier ließ sich die Fehlermeldung nicht behandeln.

Wird **VisualGeom** als Applet mit dem **appletviewer** genutzt, so ergibt sich bei dem Versuch der Nutzung von **JavaHelp** die Fehlermeldung, dass **javax/help/HelpSet** nicht gefunden würde. Mit dem Einbinden der Helpset-jar-Datei mittels **archive** in den **applet**-Tag der HTML-Datei konnten wir zwar diesen Fehler beseitigen. Dafür ließen sich dann die Hilfedateien nicht laden.

Da Applets in einer Sandbox[86] ablaufen, sind ohne Signieren Datei- und Druckoperationen nicht zugelassen. Dies muss aber getrennt für jeden Browser vorgenommen werden.
Beim Internet-Explorer 4.0 unter Windows arbeitet die Sandbox leider nicht korrekt. Alle Ein- und Ausgabeoperationen konnten von uns durchgeführt werden. Deshalb entschlossen wir uns, im Menüsystem von **VisualGeom** alle **eigentlich** im Appletmodus nicht verfügbaren Menüpunkte zu deaktivieren.

[86] Dies ist die Bezeichnung für die geschützte Umgebung, in der das Applet ohne Signieren lauffähig ist.

A.5 CD

Die CD enthält neben der Diplomarbeit im Postscript- und Winword97-Format noch weitere Dateien. Hierzu gehören:

- Die Installationsanleitungen für Java für die Betriebssysteme Windows, Linux und Solaris einschließlich spezieller Hinweise für die einzelnen Installationen und zusätzlichen Fonts für Linux, um so die Fehlermeldungen beim Programmstart beseitigen zu können
- Die Installationsanleitungen für **`VisualGeom`** für die Betriebssysteme Windows, Linux und Solaris
- Die Java-Quellcode-Dateien von **`VisualGeom`**
- Die class-Dateien von **`VisualGeom`** einschließlich der Hilfesysteme für den 2-d-Baum und den Prioritätssuchbaum in komprimierter Form getrennt für die Betriebssysteme Windows, Linux und Solaris
- Die JDK2- und JRE2-Versionen für Windows, Linux und SUN-OS
- Die jar-Datei der für das Programm benötigten Klassen von JavaHelp
- Das JavaHelp-Tool
- Die jar-Datei der Klassen zum Konvertieren in das JPEG-Format und Speichern im JPEG-Format
- Die Bedienungsanleitung aus Kapitel 10 als eigenständige Postscript-Datei
- WinZip zum Entpacken von **`VisualGeom.zip`**